Panasonic
INDUSTRY

Programmable Controller

# Programming Manual

## Control FPWIN Pro 7.7.3.0

ACGM0313V5EN

2025.02 | industry.panasonic.eu

# Copyright, liability, and warranty

## Copyright and liability



Panasonic Industry Europe pursues a policy of continuous improvement of the design and performance of its products. Therefore, we reserve the right to change the manual/product without notice. In no event will Panasonic Industry Europe be liable for direct, special, incidental, or consequential damage resulting from any defect in the product or its documentation, even if advised of the possibility of such damages.

Please direct support matters and technical questions to your local Panasonic representative.

**Panasonic Industry Europe GmbH**

Caroline-Herschel-Straße 100

85521 Ottobrunn, Germany

Tel: +49 89 45354-1000

## Limited warranty

If physical defects caused by distribution are found, Panasonic Industry Europe will replace/repair the product free of charge. Exceptions include:

- When physical defects are due to different usage/treatment of the product other than described in the manual.
- When physical defects are due to defective equipment other than the distributed product.
- When physical defects are due to modifications/repairs by someone other than Panasonic Industry Europe.
- When physical defects are due to natural disasters.

# Important symbols

One or more of the following symbols may be used in this documentation.

The following symbols are used to indicate the type of hazard.

## ⚠ DANGER

Indicates a hazardous situation which, if not avoided, will result in death or serious injury.

## ⚠ WARNING

Indicates a hazardous situation which, if not avoided, could result in death or serious injury.

## ⚠ CAUTION

Indicates a hazardous situation which, if not avoided, could result in minor or moderate injury.

**Notice**

Indicates a property damage message.

# Table of contents

**Copyright, liability, and warranty....2**

**Important symbols.... 3**

**1 Address instructions....33**

Adr_Of_Var (Returns the input or output address) .... 34

AdrLast_Of_Var (Returns the input or output address) ....36

Adr_Of_VarOffs (Returns the input or output address with offset) ....38

**2 Arithmetic instructions.... 40**

ABS (Absolute value) .... 41

ACOS (Arccosine) ....42

ADD (Add) ....44

ASIN (Arcsine) .... 46

ATAN (Arctangent) .... 48

ATAN2_YX (Returns the angle φ of the Cartesian coordinates (x,y)) ....50

ATAN2_YX_LREAL (Returns the angle φ of the Cartesian coordinates (x,y) with LREAL arguments) ....53

COS (Cosine) ....56

CRC16 (Cyclic redundancy check) .... 58

EXP (Exponent of input variable to base e) .... 60

EXPT (Raises 1st input variable by the power of the 2nd input variable) ....62

DIV (Divide) ....64

MOD (Modular arithmetic division, remainder stored in output variable) .... 66

MUL (Multiply) ....68

LIMIT (Limit value for input variable) .... 70

LN (Natural logarithm) .... 72

LOG (Logarithm to the base 10) ....74

SIN (Sine with radian input data) ....76

SQRT (Square root) ....78

SUB (Subtract) ....80

TAN (Tangent) ....82

2.22 FP instructions.... 84

FP_ADD_BCD (BCD data addition) .... 85

FP_ATAN2 (Convert coordinate data into radians) ....87

FP_COMBINE (Combine data) ..... 90
FP_COSH (Hyperbolic cosine) ..... 92
FP_DEC (Decrement) ..... 94
FP_DEC_BCD (BCD data decrement) ..... 96
FP_DIV_BCD (BCD data division) ..... 98
FP_DIV_MOD (Data division with remainder) ..... 100
FP_DIV_MOD_BCD (BCD data division with remainder) ..... 102
FP_INC (Increment) ..... 104
FP_INC_BCD (BCD data increment) ..... 106
FP_MOD_BCD (Remainder of BCD data division) ..... 108
FP_MUL_BCD (BCD data multiplication) ..... 110
FP_SINH (Hyperbolic sine) ..... 112
FP_SUB_BCD (BCD data subtraction) ..... 114
FP_TANH (Hyperbolic tangent) ..... 116
2.23 F instructions ..... 118
F40_BADD (4-digit BCD addition) ..... 119
F41_DBADD (8-digit BCD addition) ..... 121
F45_BSUB (4-digit BCD subtraction) ..... 124
F46_DBSUB (8-digit BCD subtraction) ..... 126
F52_BDIV (4-digit BCD division, destination can be specified) ..... 129
F53_DBDIV (8-digit BCD division, destination can be specified) ..... 132
F70_BCC (Block check code calculation) ..... 135
F300_BSIN (BCD type sine operation) ..... 139
F301_BCOS (BCD type cosine operation) ..... 142
F302_BTAN (BCD type tangent operation) ..... 144
F303_BASIN (BCD type arcsine operation) ..... 146
F304_BACOS (BCD type arccosine operation) ..... 148
F305_BATAN (BCD type arctangent operation) ..... 150
**3 Bistable instructions ..... 152**
R (Reset (a Boolean operand)) ..... 153
RS (Reset/set) ..... 154
RS_FUN (Reset/set) ..... 156
S (Set (a Boolean operand)) ..... 158
SR (Set/reset) ..... 159
SR_FUN (Set/reset) ..... 161

KEEP (Serves as a relay with set and reset inputs) ..... 163
SET (Set, reset output) ..... 165
**4 Bit-shift instructions ..... 167**
SHR (Shift bits to the right) ..... 168
SHL (Shift bits to the left) ..... 170
ROR (Rotate N bits to the right) ..... 172
ROL (Rotate N bits to the left) ..... 174
4.5 FP instructions ..... 176
FP_ROL_CARRY (Rotate data to the left with carry flag) ..... 177
FP_ROR_CARRY (Rotate data to the right with carry flag) ..... 179
FP_SHL_BLOCK (Shift data block to the left) ..... 181
FP_SHR_BLOCK (Shift data block to the right) ..... 183
FP_WSHL_BLOCK (Shift data block to the left) ..... 185
FP_WSHR_BLOCK (Shift data block to the right) ..... 187
4.6 F instructions ..... 189
LSR (Left shift register) ..... 190
F108_BITR (Right shift of multiple bits of 16-bit data range) ..... 192
F109_BITL (Left shift of multiple bits of 16-bit data range) ..... 195
F110_WSHR (Right shift of one word (16 bits) of 16-bit data range) ..... 198
F111_WSHL (Left shift of one word (16 bits) of 16-bit data range) ..... 200
F112_WBSR (Right shift of one hex. digit (4 bits) of 16-bit data range) ..... 202
F113_WBSL (Left shift of one hex. digit (4 bits) of 16-bit data range) ..... 204
F119_LRSR (LEFT/RIGHT shift register) ..... 206
**5 Bitwise Boolean instructions ..... 209**
AND (Logical AND operation) ..... 210
ANDN (AND NOT Connection) ..... 212
OR (OR Connection) ..... 213
ORN (OR NOT Connection) ..... 215
XOR (Exclusive OR operation) ..... 216
XORN (Exclusive OR NOT Connection) ..... 218
NOT (Bit inversion) ..... 219
5.8 FP instructions ..... 221
FP_COUNT_TRUE_BITS (Number of ON bits) ..... 222
FP_EVENTS_COUNT (Counts the number of rising edges for each bit) ..... 224
FP_EVENTS_OPERATION_TIME (Counts the operation time of each bit) ..... 226

FP_INVERT_BIT (Bit invert) ..... 228
FP_SET_BIT (Bit set) ..... 230
FP_RESET_BIT (Bit reset) ..... 232
FP_TEST_BIT (Bit test) ..... 234
5.9 F instructions ..... 236
F5_BTM (Bit data move) ..... 237
F6_DGT (Digit data move) ..... 240
F130_BTS (16-bit data bit set) ..... 242
F131_BTR (16-bit data bit reset) ..... 244
**6 Communication instructions for Ethernet clients ..... 246**
FP_FTP_GET_DATA_FORMAT (Generate data format string for FTP protocol) ..... 247
FP_HTTP_GET_DATA_FORMAT (Generate data format string for HTTP protocol) ..... 250
FP_SMTP_GET_DATA_FORMAT (Generate data format string for SMTP protocol) ..... 252
6.4 FP instructions ..... 255
FP_FTP_GET_STATUS (Get status of a single Ethernet unit using FTP transmission) ..... 256
FP_FTP_GET_STATUS_ALL (Get status of all Ethernet units using FTP transmission) ..... 258
FP_FTP_SET_CONNECTION (Server settings for FTP connection) ..... 260
FP_FTP_SET_MODE (Set FTP transfer mode) ..... 265
FP_FTP_SET_MODE_TRANSFER_LOG (Set FTP transfer mode for data recording files) ..... 270
FP_FTP_TRANSFER_CONTROL (Control of FTP transfer to a single Ethernet unit) ..... 274
FP_FTP_TRANSFER_CONTROL_ALL (Control of FTP transfer to all Ethernet units) ..... 277
FP_FTP_TRANSFER_CONTROL_LOG (Control of FTP transfer of data recording files) ..... 280
FP_FTP_TRANSFER_REQUEST (FTP transfer request) ..... 283
FP_HTTP_GET_STATUS (Get status of a single Ethernet unit using HTTP transmission) ..... 286
FP_HTTP_GET_STATUS_ALL (Get status of all Ethernet units using HTTP transmission) ..... 288
FP_HTTP_SET_CONNECTION (Server settings for HTTP connection) ..... 290
FP_HTTP_SET_MODE (Set HTTP transfer mode) ..... 295
FP_HTTP_TRANSFER_CONTROL (Control of HTTP transfer to a single Ethernet unit) ..... 299
FP_HTTP_TRANSFER_CONTROL_ALL (Control of HTTP transfer to all Ethernet units) ..... 302
FP_HTTP_TRANSFER_REQUEST (HTTP transfer request) ..... 305

FP_NTP_SET_SERVER (Set NTP server connection) ..... 308
FP_NTP_SET_SYNCHRONIZE_TIME (Set synchronization time for NTP server) ..... 311
FP_NTP_SET_TIME_ZONE (Set local time zone of NTP server) ..... 314
FP_NTP_SYNCHRONIZE (Request time synchronization from NTP server) ..... 317
FP_NTP_SET_RETRY_SETTINGS (Set retry settings for NTP server) ..... 320
FP_NTP_SET_RETRY_SETTINGS_DEFAULT (Set retry settings to default values) ..... 323
FP_SMTP_GET_EMAIL_TEXT (Get e-mail text) ..... 325
FP_SMTP_GET_STATUS (Get status of a single Ethernet unit using SMTP transmission) ..... 328
FP_SMTP_GET_STATUS_ALL (Get status of all Ethernet units using SMTP transmission) ..... 330
FP_SMTP_SET_CONNECTION (Server settings for SMTP connection) ..... 332
FP_SMTP_SET_EMAIL_TEXT (Set e-mail text) ..... 338
FP_SMTP_SET_GROUP (Set destination group) ..... 341
FP_SMTP_SET_MODE (Set SMTP transfer mode) ..... 345
FP_SMTP_SET_MODE_TRANSFER_LOG (Set SMTP transfer mode for data recording files) ..... 353
FP_SMTP_TRANSFER_CONTROL (Control of SMTP transfer to a single Ethernet unit) ..... 358
FP_SMTP_TRANSFER_CONTROL_ALL (Control of SMTP transfer to all Ethernet units) ..... 361
FP_SMTP_TRANSFER_CONTROL_LOG (Control of SMTP transfer of data recording files) ..... 364
FP_SMTP_TRANSFER_REQUEST (SMTP transfer request) ..... 367

**7 Communication instructions for Modbus/MEWTOCOL ..... 370**

7.1 Introduction to communication instructions for Modbus/MEWTOCOL ..... 371
7.2 Master function ..... 372
7.3 Slave function ..... 373
IsMasterCommunicationActive (Evaluate "IsMasterCommunicationActive" flag) ..... 375
7.5 FP instructions ..... 377
FP_MODBUS_MASTER (Write data to slave or read data from slave) ..... 378
FP_READ_FROM_SLAVE (Read data from slave) ..... 386
FP_READ_FROM_SLAVE_AREA_OFFS (Read data from slave with offset) ..... 389
FP_WRITE_TO_SLAVE (Write data to slave) ..... 393
FP_WRITE_TO_SLAVE_AREA_OFFS (Write data to slave with offset) ..... 396
7.6 F instructions ..... 400

F145_WRITE_DATA (Write data to slave) ..... 401
F145_WRITE_DATA_TYPE_OFFS (Write data to slave with type and offset) .....404
F146_READ_DATA (Read data from slave) ..... 407
F146_READ_DATA_TYPE_OFFS (Read data from slave with type and offset) .....410
F145F146_MODBUS_MASTER (Write data to slave or read data from slave) ..... 413
IsF145F146Error (Evaluate "F145/F146 error" flag) ..... 421
IsF145F146NotActive (Evaluate "F145/F146 not active" flag) ..... 423
**8 Communication instructions for networks..... 425**
8.1 Number of connections.....426
8.2 FP instructions..... 427
FP_ETHERNETIP_DATA_EXCHANGE_FB (Exchange message with EtherNet/IP unit) ...428
FP_ETHERNETIP_DATA_GET (Read data from receive buffer of EtherNet/IP unit) ..... 430
FP_ETHERNETIP_DATA_GET_BYTES (Read byte data from receive buffer of EtherNet/IP unit) ..... 432
FP_ETHERNETIP_DATA_SET (Write data to send buffer of EtherNet/IP unit) ..... 434
FP_ETHERNETIP_DATA_SET_BYTES (Write byte data to the send buffer of EtherNet/IP unit) ..... 436
FP_ETHERNETIP_GET_STATE_TABLE_ALL (Read the EtherNet/IP communication state on all nodes) ..... 438
FP_ETHERNETIP_GET_STATE_TABLE_ERROR (Find all nodes with cyclic communication errors) .....440
FP_ETHERNETIP_GET_STATE_TABLE_NODE (Find all nodes registered for cyclic communication) .....442
FP_ETHERNETIP_GET_STATE_TABLE_RUN (Find all nodes with normal cyclic communication) .....444
FP_ETHERNETIP_GET_STATE_TABLE_RUN_IDLE (Check RUN/IDLE bit for state of cyclic communication) ..... 446
FP_ETHERNETIP_GET_STATE_TABLE_STOP (Find all nodes where cyclic communication has stopped) ..... 448
FP_ETHERNETIP_GET_STATUS (Read the status of a node) ..... 450
FP_ETHERNETIP_REFRESH_INPUT (Refresh EtherNet/IP input) .....452
FP_ETHERNETIP_REFRESH_OUTPUT (Refresh EtherNet/IP output) .....455
FP_ETHERNETIP_START (Cyclic communication start request) ..... 458
FP_ETHERNETIP_STOP (Cyclic communication stop request) ..... 460
FP_MC_PROTOCOL_READ (Read data from external devices via MC protocol) ..... 462
FP_MC_PROTOCOL_WRITE (Write data to external devices via MC protocol) .....466

F145_SEND (Send data (MEWNET link)) ..... 470
F146_RECV (Receive data (MEWNET link)) ..... 472
F152_RMRD (Read data from the slave station) ..... 475
F153_RMWT (Write data into the slave station) ..... 479
**9 Communication instructions for program controlled mode ..... 482**
9.1 Sending data ..... 483
9.1.1 Writing to send buffer ..... 484
IsTransmissionDone (Evaluate "transmission done" flag) ..... 485
SendCharacters (Send characters via CPU or MCU port) ..... 487
SendCharactersAndClearString (Send characters and clear string) ..... 489
SendData (Send data via communication port or Ethernet user connection) ..... 491
9.1.6 F instructions ..... 495
F159_MTRN (Send data via CPU or MCU port) ..... 496
9.2 Receiving data ..... 499
9.2.1 Setting receive buffer for CPU ..... 501
ClearReceiveBuffer (Reset the receive buffer) ..... 505
IsReceiveBufferRead (Evaluate if the receive buffer is read) ..... 507
IsReceptionDone (Evaluate "reception done" flag) ..... 509
IsReceptionDoneByTimeOut (Evaluate "reception done" condition by timeout) ..... 511
IsRequestComPortReset (Evaluate the COM port reset requested) ..... 513
IsComPortResetDone (Evaluate if the COM port reset is done) ..... 515
ReceiveCharacters (Receive characters from CPU or MCU port) ..... 517
ReceiveData (Receive data from CPU or MCU port) ..... 519
ResetComPort (Reset a specified COM port) ..... 521
9.2.11 F instructions ..... 523
F161_MRCV (Read data from MCU port) ..... 524
9.3 Communication errors ..... 526
IsCommunicationError (Evaluate communication error flag) ..... 527
**10 Communication parameter instructions ..... 528**
10.1 Setting communication parameters ..... 529
10.1.1 Setting communication parameters in system registers ..... 530
10.1.2 Communication ports ..... 530
SetCommunicationMode (Set communication mode) ..... 532
SetProgramControlled (Set communication mode to Program controlled) ..... 534
SetMEWTOCOL (Set communication mode) ..... 535

10.1.6 DIP switch settings of Multi-Communication Unit....536
10.1.7 FP instructions....537
FP_COM_PMSET (Set communication parameters in SCU) ....538
FP_COM_SET_PARAMETER (Set communication parameters for CPU/SCU/MCU port) ....541
FP_ETHERNET_CONNECTION_CLOSE (Close an Ethernet connection) ....544
FP_ETHERNET_CONNECTION_OPEN (Open an Ethernet connection) ....546
FP_ETHERNET_CONNECTION_SET (Set an Ethernet connection) ....548
FP_IPV4_SET_ADDRESS (Set IP address using IPv4 protocol) ....555
FP_MEWNET_CLEAR_ERRORS (Clear errors in the FP7 multi-wire link unit) ....558
FP_MEWNET_W2_SET_STATION_NUMBER (Set MEWNET-W2 station number) ....560
FP_MEWNET_W_SET_PARAMETERS (Set MEWNET-W parameters) ....562
FP_MEWTOCOL_SET_MAPPING_START_FL (Set a replacement register for FL addresses This instruction sets a replacement register for FL addresses when the FP7 communicates with other FP-series controllers via MEWTOCOL.) ....564
10.1.8 F instructions....566
F159_MTRN (n_Number=16#8000) (Toggle communication mode) ....567
F159_MWRT_PARA (Set communication parameters in RUN mode) ....569
10.1.9 Data exchange with flexible network....571
FNS_InitConfigDataTable (Configure data for FP-FNS blocks) ....572
FNS_InitConfigNameTable (Configure data for FP-FNS blocks) ....575
10.2 Getting communication parameters....577
IsMasterCommunication (Evaluate "IsMasterCommunication" flag) ....578
IsPlcLink (Evaluate "PLC Link" flag) ....580
IsProgramControlled (Evaluate communication mode flag) ....581
10.2.4 FP instructions....582
FP_COM_GET_PARAMETER (Get communication parameters from CPU/SCU/MCU port) ....583
FP_COM_GET_PLCLINK_ERROR_OCCURRENCY (Get PLC link error frequency) ....585
FP_COM_GET_PLCLINK_STATIONS_PARAMETER (Get PLC link settings) ....587
FP_COM_GET_PLCLINK_STATUS (Get PLC link status) ....589
FP_COM_GET_PLCLINK_TIME_INTERVAL (Get PLC link time interval) ....591
FP_COM_GET_STATUS (Read communication port status) ....593
FP_ETHERNET_GET_STATUS (Read ET-LAN status) ....595
FP_ETHERNET_PING (Request to send PING) ....597

FP_ETHERNET_SET_TCP_DELAYED_ACK (Enable or disable the TCP delayed acknowledgement) ..... 599
FP_IPV4_GET_CONNECTION (Get connection parameters) ..... 601
FP_IPV4_GET_CONNECTION_CONFIGURED (Return the IPv4 connection parameters set by system registers) ..... 603
FP_IPV4_GET_CONNECTION_ESTABLISHED (Return the IPv4 connection parameters) ..... 605
FP_IPV4_GET_MAC (Get parameters of MAC address) ..... 607
FP_IPV6_GET_CONNECTION (Get connection parameters) ..... 609
FP_IPV6_GET_MAC (Get parameters of MAC address) ..... 611
FP_MEWNET_F_GET_NUMBER_OF_IO_POINTS (Read MEWNET-F number of I/O points) ..... 613
FP_MEWNET_F_GET_STATUS (Read MEWNET-F status information) ..... 614
FP_MEWNET_W2_GET_ERROR_HISTORY (Read MEWNET-W2 system register area for errors) ..... 615
FP_MEWNET_W2_GET_ERROR_OCCURRENCY (Get number of error occurrences on MEWNET-W2 units by error type) ..... 616
FP_MEWNET_W2_GET_NETWORK_STATUS (Get MEWNET-W2 network status information) ..... 617
FP_MEWNET_W2_GET_STATUS (Read MEWNET-W2 status information) ..... 618
FP_MEWNET_W_GET_ERROR_OCCURRENCY (Get number of error occurrences on MEWNET-W units by error type) ..... 619
FP_MEWNET_W_GET_NETWORK_STATUS (Get MEWNET-W network status information) ..... 620
FP_MEWNET_W_GET_STATUS (Read MEWNET-W status information) ..... 621
FP_MEWNET_W_GET_TIME_INTERVAL (Refresh MEWNET-W monitoring information) ..... 622
10.2.5 F instructions ..... 623
F161_MRD_PARA (Get communication parameters in RUN mode) ..... 624
F161_MRD_STATUS (Get status data in RUN mode) ..... 626
10.3 PLC Link mode ..... 627
**11 Comparison instructions ..... 628**
EQ (Equal to) ..... 629
GE (Greater than or equal to) ..... 631
GT (Greater than) ..... 633
LE (Less than or equal to) ..... 635

LT (Less than) ..... 637
NE (Not equal) ..... 639
WITHIN_LIMITS (Evaluate if a value is inside the limit) ..... 641
11.8 FP instructions ..... 643
FP_COMPARE_BLOCK (Comparison of data blocks) ..... 644
11.9 F instructions ..... 646
F64_BCMP (Comparison of data blocks) ..... 647
11.9.2 Further comparison instructions ..... 648
**12 Conversion instructions ..... 649**
TO_BOOL (Overloaded conversion to BOOL) ..... 650
TO_DINT (Overloaded conversion to DOUBLE INTEGER) ..... 651
TO_DWORD (Overloaded conversion to DOUBLE WORD) ..... 652
TO_INT (Overloaded conversion to INTEGER) ..... 653
TO_LREAL (Overloaded conversion to LREAL) ..... 654
TO_REAL (Overloaded conversion to REAL) ..... 655
TO_UDINT (Overloaded conversion to unsigned DOUBLE INTEGER) ..... 656
TO_UINT (Overloaded conversion to unsigned INTEGER) ..... 657
TO_WORD (Overloaded conversion to WORD) ..... 658
12.10 Conversion to WORD ..... 659
BOOL16_TO_WORD (BOOL16 into WORD) ..... 660
BOOLS_TO_WORD (16 variables of the data type BOOL into WORD) ..... 662
TIME_TO_WORD (TIME into WORD) ..... 664
STRING_TO_WORD (STRING (hexadecimal format) into WORD) ..... 666
STRING_TO_WORD_STEPSAVER (STRING (hexadecimal format right-justified) into WORD) ..... 667
12.11 Conversion to DWORD ..... 669
REAL_TO_DWORD (REAL into DOUBLE WORD) ..... 670
BOOL32_TO_DWORD (BOOL32 into DOUBLE WORD) ..... 671
BOOLS_TO_DWORD (32 variables of the data type BOOL into DWORD) ..... 673
TIME_TO_DWORD (TIME into DOUBLE WORD) ..... 675
STRING_TO_DWORD (STRING (hexadecimal format) into DOUBLE WORD) ..... 677
STRING_TO_DWORD_STEPSAVER (STRING (hexadecimal format right-justified) into DOUBLE WORD) ..... 679
12.12 Conversion to INT ..... 681
WORD_BCD_TO_INT (BCD value of WORD into INTEGER) ..... 682

TRUNC_TO_INT (Truncate (cut off) decimal digits of REAL or LREAL input variable, convert to INTEGER) ..... 683
TIME_TO_INT (TIME into INTEGER) ..... 685
STRING_TO_INT (STRING (decimal format) into INTEGER) ..... 687
STRING_TO_INT_STEPSAVER (STRING (decimal format right-justified) into INTEGER) ..... 689
12.13 Conversion to UINT ..... 690
WORD_BCD_TO_UINT (Binary value of WORD into unsigned INTEGER) ..... 691
TRUNC_TO_UINT (Truncate (cut off) decimal digits of REAL or LREAL input variable, convert to Unsigned INTEGER) ..... 692
STRING_TO_UINT (STRING (decimal format) into unsigned INTEGER) ..... 694
STRING_TO_UINT_STEPSAVER (STRING (decimal format right-justified) into unsigned INTEGER) ..... 696
12.14 Conversion to DINT ..... 698
DWORD_BCD_TO_DINT (Binary value of DWORD into DOUBLE INTEGER) ..... 699
TRUNC_TO_DINT (Truncate (cut off) decimal digits of REAL or LREAL input variable, convert to DOUBLE INTEGER) ..... 701
TIME_TO_DINT (TIME into DOUBLE INTEGER) ..... 703
STRING_TO_DINT (STRING (decimal format) into DOUBLE INTEGER) ..... 704
STRING_TO_DINT_STEPSAVER (STRING (decimal format right-justified) into DOUBLE INTEGER) ..... 706
12.15 Conversion to UDINT ..... 707
DWORD_BCD_TO_UDINT (Binary value of DOUBLE WORD into unsigned DOUBLE INTEGER) ..... 708
TRUNC_TO_UDINT (Truncate (cut off) decimal digits of REAL or LREAL input variable, convert to unsigned DOUBLE INTEGER) ..... 709
STRING_TO_UDINT (STRING (decimal format) into unsigned DOUBLE INTEGER) ..... 711
DATE_TO_UDINT (DATE into unsigned DOUBLE INTEGER) ..... 713
DT_TO_UDINT (DATE_AND_TIME into unsigned DOUBLE INTEGER) ..... 714
TOD_TO_UDINT (TIME_OF_DAY into unsigned DOUBLE INTEGER) ..... 715
12.16 Conversion to LREAL ..... 716
STRING_TO_LREAL (STRING into LREAL) ..... 717
TIME_TO_LREAL (TIME into LREAL) ..... 719
12.17 Conversion to REAL ..... 720
DWORD_TO_REAL (DOUBLE WORD into REAL) ..... 721
TIME_TO_REAL (TIME into REAL) ..... 722

STRING_TO_REAL (STRING into REAL) ..... 723
12.18 Conversion to TIME..... 724
WORD_TO_TIME (WORD in TIME) ..... 725
DWORD_TO_TIME (DOUBLE WORD in TIME) ..... 726
INT_TO_TIME (INTEGER into TIME) ..... 727
DINT_TO_TIME (DOUBLE INTEGER into TIME) ..... 728
LREAL_TO_TIME (LREAL into TIME) ..... 729
REAL_TO_TIME (REAL into TIME) ..... 730
12.19 Conversion to DATE_AND_TIME (DT)..... 731
UDINT_TO_DT (Unsigned DOUBLE INTEGER into DATE_AND_TIME) ..... 732
12.20 Conversion to DATE..... 733
DT_TO_DATE (DATE_AND_TIME into DATE) ..... 734
UDINT_TO_DATE (Unsigned DOUBLE INTEGER into DATE) ..... 735
12.21 Conversion to TIME_OF_DAY (TOD)..... 736
DT_TO_TOD (DATE_AND_TIME into TIME_OF_DAY) ..... 737
UDINT_TO_TOD (Unsigned DOUBLE INTEGER into TIME_OF_DAY) ..... 738
12.22 Conversion to STRING..... 739
BOOL_TO_STRING (BOOL into STRING) ..... 740
WORD_TO_STRING (WORD into STRING) ..... 742
DWORD_TO_STRING (DOUBLE WORD into STRING) ..... 745
DATE_TO_STRING (DATE into STRING) ..... 748
DT_TO_STRING (DATE_AND_TIME into STRING) ..... 749
INT_TO_STRING (INTEGER into STRING) ..... 750
INT_TO_STRING_LEFT_ALIGNED (INTEGER into left aligned STRING) ..... 753
INT_TO_STRING_LEADING_ZEROS (INTEGER into STRING) ..... 755
DINT_TO_STRING (DOUBLE INTEGER into STRING) ..... 757
DINT_TO_STRING_LEFT_ALIGNED (DOUBLE INTEGER into left-aligend STRING) ..... 759
DINT_TO_STRING_LEADING_ZEROS (DOUBLE INTEGER into STRING) ..... 761
UDINT_TO_STRING (Unsigned DOUBLE INTEGER into STRING) ..... 763
UDINT_TO_STRING_LEFT_ALIGNED (Unsigned DOUBLE INTEGER into left-aligned STRING) ..... 765
UDINT_TO_STRING_LEADING_ZEROS (Unsigned DOUBLE INTEGER into STRING) ..... 767
UINT_TO_STRING (Unsigned INTEGER into STRING) ..... 768
UINT_TO_STRING_LEFT_ALIGNED (Unsigned INTEGER into left-aligned STRING) ..... 770
UINT_TO_STRING_LEADING_ZEROS (Unsigned INTEGER into STRING) ..... 771

LREAL_TO_STRING (LREAL into STRING) ..... 772
REAL_TO_STRING (REAL into STRING) ..... 773
TIME_TO_STRING (TIME into STRING) ..... 776
IPADDR_TO_STRING (IP address into STRING) ..... 778
IPADDR_TO_STRING_NO_LEADING_ZEROS (IP address into STRING) ..... 779
ETLANADDR_TO_STRING (ETLAN address into STRING) ..... 780
ETLANADDR_TO_STRING_NO_LEADING_ZEROS (ETLAN address into STRING) ..... 781
TOD_TO_STRING (TIME_OF_DAY into STRING) ..... 782
12.23 Conversion to Special Data Types ..... 783
WORD_TO_BOOL16 (WORD into BOOL16) ..... 784
DWORD_TO_BOOL32 (DOUBLE WORD into BOOL32) ..... 785
WORD_TO_BOOLS (WORD into 16 variables of the data type BOOL) ..... 786
DWORD_TO_BOOLS (DOUBLE WORD into 32 variables of the data type BOOL) ..... 788
INT_TO_BCD_WORD (INTEGER into binary coded WORD value) ..... 790
DINT_TO_BCD_DWORD (DOUBLE INTEGER into BCD DOUBLE WORD) ..... 792
UINT_TO_BCD_WORD (Unsigned INTEGER into BCD value of WORD) ..... 794
UDINT_TO_BCD_DWORD (Unsigned DOUBLE INTEGER into BCD DOUBLE WORD) ..... 795
STRING_TO_IPADDR (STRING into IP address) ..... 796
STRING_TO_IPADDR_STEPSAVER (STRING (IP address format 00a.0bb.0cc.ddd) into DWORD) ..... 798
STRING_TO_ETLANADDR (STRING into ETLAN address) ..... 799
STRING_TO_ETLANADDR_STEPSAVER (STRING (IP address format 00a.0bb.0cc.ddd) into ETLAN address) ..... 801
12.24 FP instructions ..... 802
FP_7SEGMENT (7-segment decode) ..... 803
FP_ASCII_TO_BCD (ASCII -> BCD conversion) ..... 805
FP_ASCII_TO_BIN (ASCII -> binary conversion) ..... 809
FP_ASCII_TO_DEC (ASCII -> decimal conversion) ..... 817
FP_ASCII_TO_HEX (ASCII -> HEX conversion) ..... 820
FP_BCC (Block check character calculation) ..... 824
FP_BCD_TO_ASCII (BCD -> ASCII conversion) ..... 827
FP_BIN_TO_ASCII (Binary -> ASCII conversion) ..... 831
FP_BIN_TO_GRAY (Binary -> gray code conversion) ..... 839
FP_BITCOLUMN_TO_WORD (Bit column -> bit line conversion) ..... 841
FP_ASCII_CHECK (ASCII data check) ..... 844

FP_CRC (Cyclic redundancy check) ..... 851
FP_DEC_TO_ASCII (Decimal -> ASCII conversion) ..... 854
FP_DECODE (Decode hexadecimal -> bit state) ..... 857
FP_DEG (Angle units (radians -> degrees) conversion) ..... 862
FP_DIVIDE_BYTES (Byte distribution) ..... 864
FP_DIVIDE_DIGITS (Digit distribution) ..... 866
FP_DTBIN_TO_SEC (Binary date and time -> seconds) ..... 868
FP_ENCODE (Encode bit state -> hexadecimal) ..... 869
FP_GRAY_TO_BIN (Gray code -> binary data conversion) ..... 873
FP_HEX_TO_ASCII (HEX -> ASCII conversion) ..... 875
FP_INVERT (Data inversion (one's complement)) ..... 878
FP_RAD (Degrees -> radians conversion) ..... 880
FP_ROUND (Round up to the nearest integer) ..... 882
FP_ROUND_DOWN (Round down to the nearest integer) ..... 884
FP_UNIFY_BYTES (Byte combination) ..... 886
FP_UNIFY_DIGITS (Digit combination) ..... 888
FP_WORD_TO_BITCOLUMN (Bit line -> bit column conversion) ..... 890
12.25 F instructions ..... 893
F250_BTOA (Binary -> ASCII conversion) ..... 894
F251_ATOB (ASCII -> binary conversion) ..... 899
F252_ACHK (ASCII data check) ..... 905
F327_INT (Floating point data -> 16-bit integer data (the largest integer not exceeding the floating point data)) ..... 908
F328_DINT (Floating point data -> 32-bit integer data (the largest integer not exceeding the floating point data)) ..... 910
**13 Copy and initialize instructions ..... 912**
MOVE (Move value to specified destination) ..... 913
13.2 FP instructions ..... 915
ELC_MEMORY_READ_BLOCK (Read data block from ELC memory) ..... 916
ELC_MEMORY_WRITE_BLOCK (Write data block to ELC memory) ..... 917
FP_COPY (Copy data to memory block) ..... 918
FP_EXCHANGE (Exchange data) ..... 920
FP_MOVE2 (Move two values to a destination area) ..... 922
FP_MOVE3 (Move three values to a destination area) ..... 924
FP_MOVE_BITS (Move bit data) ..... 926
FP_MOVE_BLOCK (Move data block) ..... 928

FP_MOVE_DIGITS (Digit data move) ..... 930
FP_MOVE_INVERT (Data inversion and move) ..... 933
FP_SWAP_BYTES (Exchange higher bytes and lower bytes) ..... 935
FP_SWAP_BYTES_BLOCK (Exchange higher bytes and lower bytes) ..... 937
FP_SYSTEM_MONITOR_READ (Reads the system monitor area of the FP7) ..... 939
13.3 F instructions ..... 941
F10_BKMV_NUMBER (Block move by number) ..... 942
F10_BKMV_OFFSET (Block move to a offset from source) ..... 944
F10_BKMV_NUMBER_OFFSET (Block move by number to a offset from source) ..... 946
F17_SWAP (Higher/lower byte in 16-bit data exchange) ..... 948
F18_BXCH (16-bit blocked data exchange) ..... 950
13.4 Data transfer to and from special data registers ..... 952
13.4.1 Accessing the HSC special data registers ..... 952
13.4.2 Accessing the RTC special data registers ..... 953
**14 Counter instructions ..... 954**
CTU (Up counter) ..... 955
CTU_FUN (Up counter) ..... 958
CTD (Down counter) ..... 960
CTD_FUN (Down counter) ..... 962
CTUD (Up/down counter) ..... 964
CTUD_FUN (Up/down counter) ..... 968
CT_FB (Down counter) ..... 971
CT_FUN (Down counter) ..... 974
14.9 F instructions ..... 976
F118_UDC (UP/DOWN counter) ..... 977
**15 Data table instructions ..... 979**
15.1 FP instructions ..... 980
15.1.1 Data table analysis FP instructions ..... 981
FP_DATA_DEVIATION (Calculate deviation from data table) ..... 982
FP_DATA_SEARCH (Data search) ..... 984
FP_DATA_MAX_POS (Search maximum value in data table) ..... 986
FP_DATA_MIN_POS (Search minimum value in data table) ..... 988
FP_DATA_MEAN_SUM_DINT0 (Calculate total and mean numbers in data table of DINT numbers) ..... 990

FP_DATA_MEAN_SUM_INT (Calculate total and mean numbers in data table of INT numbers) ..... 992
FP_DATA_MEAN_SUM_REAL (Calculate total and mean numbers in data table of REAL numbers) ..... 994
FP_DATA_MEAN_SUM_UDINT0 (Calculate total and mean numbers in data table of UDINT numbers) ..... 996
FP_DATA_MEAN_SUM_UINT (Calculate total and mean numbers in data table of UINT numbers) ..... 998
15.1.2 Data table manipulation FP instructions..... 1000
FP_AVERAGE_BUFFER_DEFINE (Define buffer area for moving average and total values) ..... 1001
FP_AVERAGE_BUFFER_WRITE (Write to buffer for moving average and total values) . 1004
FP_DATA_SORT (Sort data in data table) ..... 1006
FP_DATA_READ_COMPRESS (Shift out and compress data) ..... 1008
FP_DATA_WRITE_COMPRESS (Shift in and compress data) ..... 1010
15.1.2.6 Introduction to the FIFO buffer..... 1011
FP_FIFO_DEFINE (Define the FIFO buffer area) ..... 1014
FP_FIFO_READ (Read from FIFO buffer) ..... 1016
FP_FIFO_WRITE (Write to FIFO buffer) ..... 1018
FP_LIFO_DEFINE (Define the LIFO buffer area) ..... 1020
FP_LIFO_READ (Read from LIFO buffer) ..... 1022
FP_LIFO_WRITE (Write to LIFO buffer) ..... 1024
15.2 F instructions..... 1026
15.2.1 Data table analysis F instructions..... 1027
F96_SRC (Table data search (16-bit search)) ..... 1028
F97_DSRC (32-bit table data search) ..... 1030
F270_MAX (Maximum value search in 16-bit data table) ..... 1032
F271_DMAX (Maximum value search in 32-bit data table) ..... 1034
F350_FMAX (Maximum value search in real number data table (floating point data)) ..... 1036
F272_MIN (Minimum value search in 16-bit data table) ..... 1038
F273_DMIN (Minimum value search in 32-bit data table) ..... 1040
F351_FMIN (Minimum value search in real number data table (floating point data)) ..... 1042
F275_MEAN (Total and mean numbers calculation in 16-bit data table) ..... 1044
F276_DMEAN (Total and mean numbers calculation in 32-bit data table) ..... 1046
F352_FMEAN (Total and mean numbers calculation in floating point data table) ..... 1048
15.2.2 Data table manipulation F instructions..... 1050

F115_FIFT (FIFO buffer area definition) ..... 1051
F116_FIFR (Read from FIFO buffer) ..... 1056
F117_FIFW (Write to FIFO buffer) ..... 1061
F277_SORT (Sort data in 16-bit data table) ..... 1066
F278_DSORT (Sort data in 32-bit data table (in smaller or larger number order)) ..... 1068
F353_FSORT (Sort data in real number data table (floating point data table)) ..... 1070
**16 Date and time instructions ..... 1073**
ADD_DT_TIME (Add TIME to DATE_AND_TIME) ..... 1074
ADD_TOD_TIME (Add TIME to TIME_OF_DAY) ..... 1075
CONCAT_DATE_INT (Concatenate INT values to form a date) ..... 1076
CONCAT_DATE_TOD (Concatenate date and time of day) ..... 1078
CONCAT_DT_INT (Concatenate INT values to form date and time) ..... 1079
CONCAT_TOD_INT (Concatenate INT values to form the time of day) ..... 1081
DAY_OF_WEEK0 (Return the day of the week) ..... 1083
DTBCD_TO_DT (DTBCD to DATE_AND_TIME) ..... 1084
DTBIN_TO_DT (Date and time in binary format to DATE_AND_TIME) ..... 1085
DT_TO_DTBCD (DATE_AND_TIME to date and time data in BCD format) ..... 1087
DT_TO_DTBIN (DATE_AND_TIME to date and time data in binary format) ..... 1088
GET_RTC_DT (Read the real-time clock) ..... 1090
GET_RTC_DTBCD (Read the Real-Time Clock using DTBCD) ..... 1091
GET_RTC_DTBIN (Read the Real-Time Clock using DTBIN) ..... 1092
SET_RTC_DTBCD (Set the Real-Time Clock using DTBCD) ..... 1093
SET_RTC_DTBIN (Set the Real-Time Clock using DTBIN) ..... 1094
IS_VALID_DATE_INT (Check whether DATE is valid) ..... 1095
IS_VALID_DT_INT (Check whether DATE_AND_TIME is valid) ..... 1097
IS_VALID_TOD_INT (Check whether the TIME_OF_DAY is valid) ..... 1099
SET_RTC_DT (Set the real-time clock) ..... 1101
SPLIT_DATE_INT (Split DATE into INT values) ..... 1102
SPLIT_DT_INT (Split DATE_AND_TIME into INT values) ..... 1104
SPLIT_TOD_INT (Split TIME_OF_DAY into INT values) ..... 1106
SUB_DATE_DATE (Subtract DATE from DATE) ..... 1108
SUB_DT_DT (Subtract DATE_AND_TIME from DATE_AND_TIME) ..... 1110
SUB_DT_TIME (Subtract TIME from DATE_AND_TIME) ..... 1112
SUB_TOD_TIME (Subtract TIME from TIME_OF_DAY) ..... 1113
SUB_TOD_TOD (Subtract TIME_OF_DAY from TIME_OF_DAY) ..... 1114

16.29 F instructions....1115
F230_DTBCD_TO_SEC (Time data conversion into seconds) ....1116
F231_SEC_TO_DTBCD (Conversion of seconds into time data) ....1117
**17 Edge detection instructions....1119**
R_TRIG (Rising edge trigger) ....1120
R_TRIG_FUN (Rising edge trigger) ....1122
F_TRIG (Detecting a falling edge) ....1123
F_TRIG_FUN (Detecting a falling edge) ....1125
17.5 FP instructions....1126
FP_DETECT_CHANGE (Detect changes in data) ....1127
17.6 F instructions....1129
ALT (Alternative out) ....1130
DF (Rising edge differential) ....1132
DFI (Rising edge differential (initial execution type)) ....1134
DFN (Falling edge differential) ....1136
**18 FP-e display instructions....1138**
F180_SCR (Screen display instruction) ....1139
F180_SCR_DUT (Configuring the display of the FP-e) ....1140
F181_DSP (Screen change instruction) ....1148
**19 GT panel instructions....1151**
GT_ActivateScreen (Control the GT panel screen) ....1152
GT_ChangeBacklightBrightness (Changes the backlight brightness of a GT panel) ....1154
**20 High-speed counter instructions....1156**
20.1 Introduction....1157
20.2 F instructions....1163
20.2.1 Writing the high-speed counter control code....1163
Cancelling high-speed counter instructions (bit 3)....1163
Enabling/disabling the reset input (hardware reset) of the high-speed counter (bit 2)....1164
Enabling/disabling counting operations (bit 1)....1164
Resetting the elapsed value (software reset) of the high-speed counter to 0 (bit 0)....1165
Description for FP-Sigma, FP-X, FP0R....1165
Description for FP0, FP-e....1166
20.2.2 High-speed counter: writing and reading the elapsed value....1168
F165_HighSpeedCounter_Cam (Cam control) ....1171
F166_HighSpeedCounter_Set (Target value match ON (high-speed counter)) ....1185

F167_HighSpeedCounter_Reset (Target value match OFF (high-speed counter)) ..... 1189
F178_HighSpeedCounter_Measure (Input pulse measurement) ..... 1193
20.3 Tool instructions for high-speed counter available for (FPΣ, FPX, FP0R, FPe, FP0)..... 1196
20.3.1 High-speed counter control instructions..... 1196
HscControl_CountingDisable (Disables counting on high-speed counter channel) ..... 1197
HscControl_CountingEnable (Enables counting on high-speed counter channel) ..... 1199
HscControl_ElapsedValueContinue (Continues counting after reset) ..... 1201
HscControl_ElapsedValueReset (Sets elapsed value to 0) ..... 1203
HscControl_HscInstructionClear (Clears high-speed counter instruction) ..... 1205
HscControl_ResetInputDisable (Disables reset input) ..... 1207
HscControl_ResetInputEnable (Enables reset input) ..... 1209
HscControl_SetDefaults (Sets defaults for high-speed counter channel) ..... 1211
HscControl_WriteElapsedValue (Writes elapsed value into high-speed counter channel) ..... 1213
20.3.2 High-speed counter information instructions..... 1214
HscInfo_GetControlCode (Returns control code of high-speed counter channel) ..... 1215
HscInfo_GetCurrentSpeed (Returns current speed of high-speed counter channel) ..... 1217
HscInfo_IsActive (Checks if high-speed counter is active) ..... 1219
HscInfo_IsChannelEnabled (Checks if high-speed counter channel is enabled) ..... 1221
HscInfo_IsCountingDisabled (Checks if counting is disabled) ..... 1223
HscInfo_IsElapsedValueReset (Checks if elapsed value is set to 0) ..... 1225
HscInfo_IsResetInputDisabled (Checks if reset input is disabled) ..... 1227
HscInfo_ReadElapsedValue (Reads elapsed value from high-speed counter channel) ... 1229
HscInfo_ReadTargetValue (Reads target value from high-speed counter channel) ..... 1231
20.3.3 High-speed counter target value match control..... 1232
Hsc_TargetValueMatch_Reset (Target value match OFF (high-speed counter)) ..... 1233
Hsc_TargetValueMatch_Set (Target value match ON (high-speed couter)) ..... 1236
**21 Input, output and unit access instructions..... 1239**
ExpansionUnitNumberToIOWordOffset_FPX_FP0 (Function to calculate the I/O offset of an analog expansion unit connected to the CPU via an adapter.) ..... 1240
ExpansionUnitNumberToIOWordOffset_FP0 (Function to calculate the I/O offset of an analog expansion unit connected directly to the CPU (without adapter).) ..... 1242
Unit_AnalogInOut_FP0R_A21 (Function block to write to and read from an FP0R-A21 unit.) ..... 1244
Unit_AnalogInOut_FP0R_A42 (Function block to write to and read from an FP0R-A42 unit.) ..... 1249

Unit_AnalogInOut_FP0_A21 (Function to write to and read from an FP0-A21 unit.) ..... 1255
Unit_AnalogInput_FP0R_AD4 (Function block to read from an FP0R-AD4 unit.) ..... 1261
Unit_AnalogInput_FP0R_AD8 (Function block to read from an FP0R-AD8 unit.) ..... 1265
Unit_AnalogInput_FP0_A80 (Function block to read from an FP0-A80 unit.) ..... 1270
Unit_AnalogInput_FP0_RTD_INT (Function block to read from an FP0-RTD6 unit (converted digital values are of data type INT).) ..... 1275
Unit_AnalogInput_FP0_RTD_REAL (Function block to read from an FP0-RTD6 unit (converted digital values are of data type REAL)) ..... 1281
Unit_AnalogInput_FP0_TC4_TC8 (Function block to read from an FP0-TC4 or FP0-TC8 unit.) ..... 1287
Unit_AnalogOutput_FP0R_DA4 (Function block to write to an FP0R-DA4 unit.) ..... 1291
Unit_AnalogOutput_FP0_A04I ( Function to write to an FP0-A04I unit.) ..... 1296
Unit_AnalogOutput_FP0_A04V (Function to write to an FP0-A04V unit.) ..... 1299
Unit_AnalogInOut_FPG_A44 (Function to write to and read from FPG-A44 unit) ..... 1302
Unit_AnalogInput_FP7_AD4H (Function to read from an FP7-AD4H unit.) ..... 1305
Unit_AnalogInput_FP7_AD8 (Function to read from an FP7-AD8 unit.) ..... 1306
Unit_AnalogInput_FP7_Control (Function to write control values to and read status values from an FP7 analog unit.) ..... 1307
Unit_AnalogInput_FP7_RTD8 (Function to read from an FP7-RT8D unit.) ..... 1308
Unit_AnalogInput_FP7_TC8 (Function to read from an FP7-TC8 unit.) ..... 1310
Unit_AnalogOutput_FP7_DA4H (Function block to write to an FP7-DA4H unit.) ..... 1312
21.22 FP instructions ..... 1313
FP_CLEAR_UNIT_ERROR (Clear error/warning of units) ..... 1314
FP_DIRECT_INPUT (Direct input refresh) ..... 1316
FP_DIRECT_IN (Read direct input flag) ..... 1318
FP_DIRECT_OUTPUT (Direct output refresh) ..... 1320
FP_DIRECT_OUT (Write direct output flag) ..... 1322
FP_GET_IO_START_OFFSET (Get offset of I/O address) ..... 1324
FP_GET_UNIT_ID (returns the expansion unit ID of a specified slot) ..... 1326
FP_GET_UNIT_OFFSETS1 (Calculate the I/O offset of an expansion unit) ..... 1328
21.23 F instructions ..... 1331
F143_IORF (Partial I/O update) ..... 1332
F150_READ (Data read from intelligent units) ..... 1334
F151_WRT (Write into memory of intelligent units) ..... 1337
F147_PR (Parallel printout) ..... 1340
**22 Memory device instructions ..... 1343**

ReadDataFromFileRegisterBank (Read data from file register bank 1 or 2) .....1344
WriteDataToFileRegisterBank (Write data to file register bank 1 or 2) .....1346
22.3 FP instructions..... 1348
22.3.1 Introduction to SD card instructions..... 1348
Using SD card instructions with FP-XH Ethernet type PLCs..... 1348
Operation of instructions.....1348
Flag operation.....1349
List of error codes..... 1349
How to specify directory and file names in an SD card..... 1351
SD card specifications.....1351
FP_SD_COPY_FILE (Copy file or directory on SD card) ..... 1352
FP_SD_CREATE_DIR (Create directory on SD card) ..... 1354
FP_SD_DELETE_DIR (Delete directory on SD card) .....1356
FP_SD_DELETE_DIR_WITH_FILES (Delete directory including files on SD card) ..... 1358
FP_SD_DELETE_FILE (Delete file from SD card) ..... 1360
FP_SD_GET_FILE_STATUS (Return the properties of a specified file on the SD card) ...1362
FP_SD_GET_FREE_KBYTES (Return the number of free kBytes on the SD card) ..... 1364
FP_SD_MOVE_FILE (Move file on SD card) ..... 1366
FP_SD_READ (Read data from SD card) ..... 1368
FP_SD_READ_BIN (Read binary data from SD card) ..... 1371
FP_SD_READ_LINE (Read one line from a specified file from SD card) .....1374
FP_SD_RENAME_FILE (Rename file on SD card) .....1376
FP_SD_WRITE (Write data to SD card) .....1378
FP_SD_WRITE_BIN (Write binary data to SD card) ..... 1381
FP_SD_WRITE_LINE (Write one line to a specified file on SD card) .....1384
FP_LOGTRACE_SAMPLE (Perform data recording triggered by instruction) .....1386
FP_LOGTRACE_START (Start data recording) .....1388
FP_LOGTRACE_STOP (Stop data recording) .....1390
FP_OPERATION_RECORDING_CLEAR (Clear operation record) .....1392
FP_OPERATION_RECORDING_START (Start operation recording) .....1394
FP_OPERATION_RECORDING_STOP (Stop operation recording) .....1396
FP_OPERATION_RECORDING_WRITE_TO_SD (Write operation record to SD card) ...1398
22.4 F instructions.....1400
F12_EPRD (EEPROM read from memory) .....1401
F12_ICRD (IC card extended memory read) .....1404

F13_ICWT (IC card extended memory write) ..... 1406
F14_PGRD (Program read from IC card) ..... 1408
P13_EPWT (EEPROM write to memory) ..... 1409
F155_SMPL (Transfer sampling data) ..... 1412
F156_STRG (Set sampling trigger) ..... 1413
**23 Pointer instructions ..... 1414**
AdrDT_Of_Offs (Returns the DT address of the input or output with offset) ..... 1415
AdrDT_Of_Offs32 (Returns the DT address of the input or output with 32-bit offset) ..... 1418
AdrFL_Of_Offs (Returns the FL address of the input or output with offset) ..... 1420
AdrFL_Of_Offs32 (Returns the FL address of the input or output with 32-bit offset) ..... 1422
AreaOffs32_ToVar (Copies the content of an address to a variable with 32-bit offset) ..... 1424
AreaOffs_ToVar (Copies the content of an address to a variable with 16-bit offset) ..... 1428
GetPointer (provides pointer information) ..... 1432
GetPointer32 (Provides pointer information) ..... 1435
Is_AreaDT (Yields TRUE if the memory area of a variable is a DT area) ..... 1437
Is_AreaFL (Yields TRUE if the memory area of a variable is a FL area) ..... 1438
Var_ToAreaOffs (Copies a variable value to an address area) ..... 1439
Var_ToAreaOffs32 (Copies a variable value to an address area) ..... 1443
**24 Process control instructions ..... 1447**
CAL (Unconditional Call) ..... 1448
CALC (Conditional Call) ..... 1449
CALCN (Conditional Call NOT) ..... 1450
JMP (Unconditional Jump) ..... 1451
JMPC (Conditional Jump) ..... 1453
JMPCN (Conditional Jump NOT) ..... 1455
RET (Unconditional Return from Function-Block) ..... 1457
RETC (Conditional Return from Function-Block) ..... 1458
RETCN (Conditional Return NOT from Function Block) ..... 1459
24.10 FP instructions ..... 1460
24.10.1 Explanation of the operation of the PID instructions ..... 1460
PID_FB (PID processing instruction) ..... 1465
PID_FB_DUT (PID processing instruction) ..... 1468
FP_PID_BASIC (PID processing) ..... 1470
FP_PID (PID processing with optional PWM output) ..... 1472
FP_RAMP (Ramp output) ..... 1475

24.11 F instructions....1478
F355_PID_DUT (PID processing instruction) ....1479
Example....1479
F356_PID_PWM (PID processing with optional PWM output) ....1482
**25 Program execution control instructions....1486**
BRK (Break) ....1487
FP_END_SCAN (Conditional end) ....1488
ICTL (Interrupt control) ....1490
25.4 FP instructions....1493
FP_INTERRUPT_ACTIVATE (Control the activation of interrupt programs for one unit) ..1494
FP_INTERRUPT_CLEAR_REQUESTS (Clear interrupt programs for one unit) ....1496
FP_INTERRUPT_DISABLE (Disable interrupt programs) ....1499
FP_INTERRUPT_ENABLE (Enable interrupt programs) ....1501
25.5 F instructions....1503
F19_SJP (Indirect jump to label) ....1504
**26 Pulse output instructions....1506**
26.1 Introduction....1507
26.2 Table operation mode (FP7, FP-XH, FP0H)....1512
26.2.1 FP instructions....1513
FP_POS_UNIT_GET_ERROR (Get error or warning in positioning unit) ....1514
FP_POS_UNIT_GET_STATUS (Get axis status of positioning unit) ....1516
FP_POS_UNIT_SET_TABLE (Set positioning table) ....1518
26.2.2 F instructions....1521
F380_Positioning_Start (Positioning table start) ....1522
F381_Positioning_Jog (JOG operation start) ....1524
F382_Positioning_Home (Home return) ....1526
F383_Positioning_StartMultiple (Simultaneous start of multiple positioning tables) ....1528
F384_Positioning_ReadData (Read positioning parameters) ....1530
F385_Positioning_WriteData (Write positioning parameters) ....1532
F385_Positioning_WriteData_Backup (Write positioning parameters with backup) ....1535
F386_Positioning_SetTable (Set positioning table (FP0H, FP-XH not M4/M8 type)) ....1538
F387_Positioning_GetStatus (Get axis status of positioning unit (FP0H, FP-XH not M4/M8 type)) ....1539
F388_Positioning_GetError (Get error or warning in positioning unit (FP0H, FP-XH not M4/M8 type)) ....1541

F389_Positioning_ClearError (Clear error or warning of positioning unit (FP0H, FP-XH not M4/M8 type)) ..... 1542
26.3 Instruction operation mode (FP0, FP0H, FP-e, FP-Sigma, FP-XH)..... 1543
26.3.1 Introduction to pulse output instructions..... 1543
26.3.2 Writing the pulse output control code..... 1546
Setting/resetting near home input..... 1547
Enabling/disabling counting operations..... 1548
Resetting the elapsed value (software reset) of the high-speed counter..... 1549
Cancelling high-speed counter and position control instructions (FP0R only)..... 1549
Description for FP-Sigma..... 1549
Description for FP-X..... 1550
Description for FP0R..... 1551
Description for FP0, FP-e..... 1552
26.3.3 Pulse output: writing and reading the elapsed value..... 1554
26.3.4 F instructions..... 1557
F166_PulseOutput_Set (Target value match ON (pulse output)) ..... 1558
F167_PulseOutput_Reset (Target value match OFF (pulse output)) ..... 1561
F168_PulseOutput_Trapezoidal (Trapezoidal control) ..... 1564
F168_PulseOutput_Home (Home return) ..... 1569
F170_PulseOutput_PWM (PWM output) ..... 1574
F171_PulseOutput_Trapezoidal (Trapezoidal Control) ..... 1577
F171_PulseOutput_Home (Home return) ..... 1583
F171_PulseOutput_Jog_Positioning (JOG operation and positioning) ..... 1588
F169_PulseOutput_Jog (JOG operation) ..... 1594
F172_PulseOutput_Jog (JOG operation) ..... 1598
F173_PulseOutput_PWM (Pulse output instruction with channel specification (PWM output)) ..... 1604
F173_PulseOutput_PWM_Hz (Pulse output instruction with channel specification (PWM output)) ..... 1607
F174_PulseOutput_DataTable (Data table control) ..... 1609
F175_PulseOutput_Linear (Linear interpolation) ..... 1613
F176_PulseOutput_Center (Circular interpolation (center position)) ..... 1618
F176_PulseOutput_Pass (Circular interpolation (pass position)) ..... 1623
F177_PulseOutput_Home (Home return) ..... 1628
26.3.5 Tool instructions for pulse output..... 1632
26.3.5.1 Channel configuration DUT for all PLCs and all pulse output instructions..... 1634

26.3.5.2 Pulse output function blocks....1637
PulseOutput_Center_FB (Circular interpolation (center position)) ....1638
PulseOutput_Home_FB (Home return) ....1642
PulseOutput_Jog_FB (JOG operation) .... 1646
PulseOutput_Jog_Positioning0_FB (JOG operation and positioning) ....1649
PulseOutput_Jog_Positioning1_FB (JOG operation and positioning) ....1652
PulseOutput_Jog_TargetValue_FB (JOG operation with target value) .... 1655
PulseOutput_Linear_FB (Linear interpolation) .... 1658
PulseOutput_Pass_FB (Circular interpolation (pass position)) .... 1662
PulseOutput_Trapezoidal_FB (Trapezoidal control) .... 1666
26.3.5.3 Pulse control instructions....1669
PulseControl_CountingDisable (Disables counting on pulse output channel) .... 1670
PulseControl_CountingEnable (Enables counting on pulse output channel) .... 1672
PulseControl_DeceleratedStop (Performs decelerated stop) .... 1673
PulseControl_ElapsedValueContinue (Continues pulse counting after reset) ....1675
PulseControl_ElapsedValueReset (Sets elapsed value to 0) .... 1677
PulseControl_JogPositionControl (Starts position control) ....1679
PulseControl_NearHome (Starts deceleration when near home) .... 1681
PulseControl_PulseOutputContinue (Continues pulse output) .... 1683
PulseControl_PulseOutputStop (Stops pulse output) .... 1684
PulseControl_SetDefaults (Sets defaults for pulse output channel) .... 1685
PulseControl_WriteElapsedValue (Writes elapsed value into pulse output channel) ....1687
PulseControl_TargetValueMatchClear (Clears target value match control) .... 1689
26.3.5.4 Pulse information instructions....1691
PulseInfo_GetControlCode (Returns control code of pulse output channel) ....1692
PulseInfo_GetCurrentSpeed (Returns current pulse output frequency) ....1694
PulseInfo_IsActive (Check if pulse output is active) .... 1696
PulseInfo_IsChannelEnabled (Checks if pulse output channel is enabled) .... 1697
PulseInfo_IsCountingDisabled (Checks if pulse counting is disabled) ....1699
PulseInfo_IsElapsedValueReset (Checks if elapsed value is set to 0) .... 1701
PulseInfo_IsHomeInputTrue (Checks if home input is TRUE) .... 1703
PulseInfo_IsPulseOutputStopped (Checks if pulse output has stopped) .... 1705
PulseInfo_IsTargetValueMatchActive (Checks if target value match control is active) .... 1707

PulseInfo_ReadAccelerationForbiddenAreaStartingPosition (Read acceleration forbidden area starting position) ..... 1709
PulseInfo_ReadCorrectedFinalSpeed (Reads corrected value of final speed) ... 1711
PulseInfo_ReadCorrectedInitialSpeed (Reads corrected value of initial speed) . 1713
PulseInfo_ReadElapsedValue (Reads elapsed value from pulse output channel) ..... 1715
PulseInfo_ReadTargetValue (Reads target value from pulse output channel) .... 1717
PulseInfo_ReadTargetValueMatchValue (Reads output control target value from pulse output channel) ..... 1719
26.3.5.5 Pulse output target value match control..... 1721
Pulse_TargetValueMatch_Reset (Target value match OFF (pulse output)) ..... 1722
Pulse_TargetValueMatch_Set (Target value match ON (pulse output)) ..... 1725
**27 Selection instructions..... 1728**
MAX (Maximum value) ..... 1729
MIN (Minimum value) ..... 1731
MUX (Select value from multiple channels) ..... 1733
SEL (Select value from one of two channels) ..... 1735
**28 SFC control instructions..... 1737**
28.1 Instructions that control all SFC programs simultaneously..... 1738
StartStopAllSfcs (Stop and restart all SFC programs) ..... 1739
StartStopAllSfcsAndInitData (Stop and restart all SFC programs) ..... 1741
28.2 A function that reveals the status of all SFCs..... 1743
AllSfcsStopped (Indicates whether all SFCs were stopped) ..... 1744
28.3 Instructions that control a specific SFC..... 1745
StartStopSfc (Stop and restart a specific SFC program) ..... 1746
StartStopSfcAndInitData (Stop and restart a specific SFC program) ..... 1748
ControlSfc (Control a specific SFC program) ..... 1750
ControlSfcAndData (Control a specific SFC program) ..... 1752
ActivateStepsOfStoppedSfc (Continue a stopped SFC program at any position) ..... 1754
28.4 Instructions that reveal the statuses of a specific SFC..... 1756
SfcStopped (Indicates whether a specific SFC program was stopped) ..... 1757
SfcTransitionsInhibited (Indicates whether the transitions of a specific SFC program are locked) ..... 1758
SfcRunning (Indicates whether a certain SFC program is running) ..... 1759
SfcOutputsReset (Indicates whether the outputs of a SFC program have been reset) ..... 1760
**29 Signal processing instructions..... 1761**

SCALE_INT (Scale INTEGER data) ..... 1762
SCALE_INT_UINT (Scale INTEGER data to unsigned INTEGER data) ..... 1764
SCALE_REAL (Scale REAL data) ..... 1766
SCALE_UINT (Scale UINT data) ..... 1768
SCALE_UINT_INT (Scale UINT data to INT data) ..... 1770
SmoothSignal_INT (Smooth INT signals) ..... 1772
SmoothSignal_REAL (Smooth REAL signals) ..... 1774
SmoothSignal_UINT (Smooth UINT signals) ..... 1776
29.9 FP instructions ..... 1778
FP_DEBOUNCE (Debounce data by filtering bits over a specified time) ..... 1779
FP_SCALE (Perform linear interpolation of discrete values) ..... 1782
FP_BAND (Compare with deadband) ..... 1786
FP_ZONE (Add offset to input value) ..... 1789
29.10 F instructions ..... 1792
F282_SCAL (Linear interpolation of discrete INT values) ..... 1793
F283_DSCAL (Linear interpolation of discrete DINT values) ..... 1797
F354_FSCAL (Linear interpolation of discrete REAL values) ..... 1801
**30 Size information instructions ..... 1804**
Elem_OfArray1D (Number of elements in an array) ..... 1805
Elem_OfArray2D (Number of elements in an array) ..... 1807
Elem_OfArray3D (Number of elements in an array) ..... 1809
GetDataTypeInfo (Read the description of a structured variable) ..... 1811
Size_Of_Var (Returns the size of a variable) ..... 1813
**31 Special instructions ..... 1814**
31.1 FP instructions ..... 1815
FP_RESET_WATCHDOG (Reset the watchdog timer) ..... 1816
FP_SET_ERROR (Set/reset self-diagnostic error) ..... 1817
31.2 F instructions ..... 1819
F140_STC (Carry-flag set) ..... 1820
F141_CLC (Carry-flag reset) ..... 1821
F142_WDT (Watchdog timer update) ..... 1822
F149_MSG (Message display) ..... 1824
**32 String instructions ..... 1826**
CONCAT (Concatenate (attach) a string) ..... 1827
DELETE (Delete characters from a string) ..... 1829

FIND (Find string's position) ..... 1831
FIND_AFTER_POS (Find string's position) ..... 1833
INSERT (Insert characters) ..... 1835
LEFT (Copy characters from the left) ..... 1837
LEN (String length) ..... 1839
MAX_LEN (Returns the maximum string length) ..... 1841
MID (Copy characters from a middle position) ..... 1843
REPLACE (Replaces characters) ..... 1845
RIGHT (Copy characters from the right) ..... 1847
SET_LEN (Set string length) ..... 1849
32.13 FP instructions ..... 1851
FP_FORMAT_STRING (Write formatted data into a result string) ..... 1852
**33 System register instructions ..... 1859**
SYS1 (Change PLC system setting) ..... 1860
**SYS1** Communication condition setting for the COM ports of the CPU ..... 1861
**SYS1** Password setting ..... 1866
**SYS1** Interrupt setting ..... 1868
**SYS1** PLC link time setting ..... 1870
**SYS1** Change high-speed counter operation mode ..... 1873
**SYS1** RS485 response time control ..... 1875
SYS2 (Change system register settings for PC link area) ..... 1878
**34 Timer instructions ..... 1881**
ADD_TIME (Add TIME) ..... 1882
CONCAT_TIME_INT (Concatenate INT values to form a time) ..... 1884
DIV_TIME_DINT (Divide TIME by DOUBLE INTEGER) ..... 1886
DIV_TIME_INT (Divide TIME by INTEGER) ..... 1888
DIV_TIME_REAL (Divide TIME by REAL) ..... 1890
MUL_TIME_DINT (Multiply TIME by DOUBLE INTEGER) ..... 1892
MUL_TIME_INT (Multiply TIME by INTEGER) ..... 1894
MUL_TIME_REAL (Multiply TIME by REAL) ..... 1896
SPLIT_TIME_INT (Split TIME into INT values) ..... 1898
SUB_TIME (Subtract TIME) ..... 1900
TM_100ms_FB (Timer for 100ms intervals (0 to 3276.7s)) ..... 1902
TM_100ms_FUN (Timer for 100ms intervals (0 to 3276.7s)) ..... 1905
TM_10ms_FB (Timer for 10ms intervals (0 to 327.67s)) ..... 1907

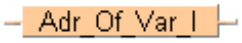
TM_10ms_FUN (Timer for 10ms intervals (0 to 327.67s)) ..... 1910
TM_1ms_FB (Timer for 1ms intervals (0 to 32.767s)) ..... 1912
TM_1ms_FUN (Timer for 1ms intervals (0 to 32.767s)) ..... 1915
TM_1s_FB (Timer for 1s intervals (0 to 32767s)) ..... 1917
TM_1s_FUN (Timer for 1s intervals (0 to 32767s)) ..... 1920
TOF (Timer with switch-off delay) ..... 1922
TOF_FUN (Timer with switch-off delay) ..... 1924
TON (Timer with switch-on delay) ..... 1926
TON_FUN (Timer with switch-on delay) ..... 1929
TP (Timer with defined period) ..... 1931
TP_FUN (Timer with defined period) ..... 1934
34.25 F instructions..... 1936
F137_STMR (Timer 16 bit) ..... 1937
F183_DSTM (Timer 32-bit) ..... 1939
**35 Appendix..... 1941**
35.1 FP7 instructions..... 1941
35.2 Advantages of FP instructions..... 1948
35.3 Advantages of IEC instructions..... 1949
35.4 7-Segment conversion table..... 1952
35.5 BCD data..... 1953
Handling BCD data in the PLC..... 1953
35.6 Decimal to binary/BCD/gray code table..... 1956
35.7 Additional examples for Control FPWIN Pro7..... 1957
35.8 Error codes..... 1959
35.8.1 Table of syntax check error..... 1959
35.8.2 Table of self-diagnostic errors..... 1960
35.8.3 Table of communication check error..... 1966
35.8.4 Table of Ethernet communication error codes..... 1966
35.8.5 Table of Modbus/MEWTOCOL communication error codes (FP7 only)..... 1970
**36 Record of changes..... 1972**
**Index..... 1973**

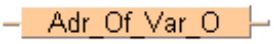
# 1 Address instructions

# Adr_Of_Var

Returns the input or output address

This function returns the address of a variable at the input or output of a non-overloaded system instruction.




## Parameters

Input

**Unnamed input** (ANY)

This pin must be connected to the input/output of a basic function for which the data type INT, WORD is allowed
Yields the 16-bit starting address of the input/output variables

Output

**Unnamed output** (ANY_IN_UNITS_OF_WORDS)

Input/output variable for which the 16-bit starting address is needed

## Remarks

- The 16-bit starting address is yielded at input/output **Adr** based on the variables at input/output **Var**. This input/output has to be directly attached to the 16-bit input or output of a non-overloaded system instruction.
- Only for LD and FBD Editors: Use “Input instruction” or “Output instruction” from the “Instructions” pane to insert the instruction required into the programming window.

## Example

### DUT

Under “DUTs” a structured data type is assigned in which the structure’s various, non-boolean variables are declared.

DUT_NonBoolean [DUT]

| | Identifier | Type | Initial |
|---|---|---|---|
| 0 | Int1 | INT | 0 |
| 1 | ArrayOfInt1 | ARRAY [0..2] OF INT | [3(0)] |
| 2 | Word1 | WORD | 0 |
| 3 | ArrayOfDint1 | ARRAY [0..2] OF DINT | [3(0)] |
| 4 | Dword1 | DWORD | 0 |
| 5 | Dint1 | DINT | 0 |

## POU header

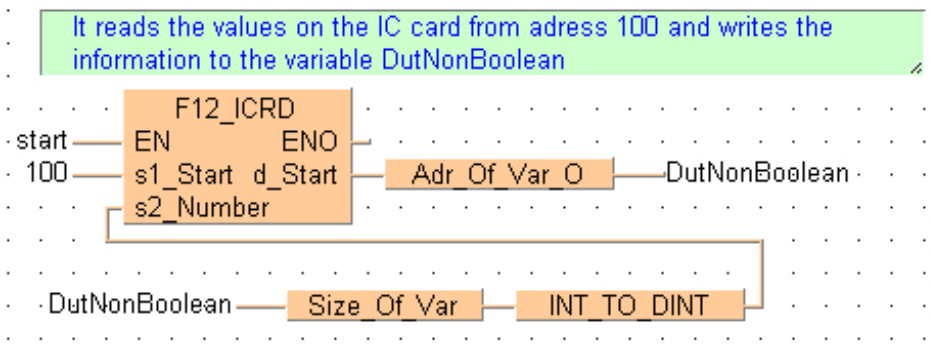
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activation of the function |
| 1 | VAR | DutNonBoolean | Dut_NonBoolean | | structured data type |

Here the variable **DutNonBoolean** of the data type assigned in the above DUT is declared. Assigning elements of the variable **DutNonBoolean** with values was not done in the POU header or body because the values of the variable **DutNonBoolean** are overwritten after the function **F12_ICRD** is executed.

When the variable **start** is set to TRUE, the function **F12_ICRD** is carried out. The function reads values on the IC card beginning with address 100 and writes the information to the variable **DutNonBoolean**. Do not forget that the IC card has to be appropriately formatted via the menu “Online” > “IC memory card manager...” and that, if necessary, values beginning at address 100 should be available on the IC card.

## LD body



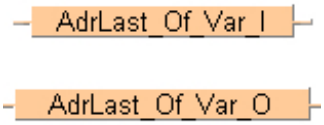

Further example projects (directory "Samples" of FPWIN Pro installation):

- Read, write IC card
- Read, write shared memory

# AdrLast_Of_Var

Returns the input or output address

This function returns the address of a variable at the input or output of a basic function.




## Parameters

Input

**AdrLast** (ANY)

This pin must be connected to the input/output of a basic function for which the data type INT, WORD is allowed

Yields the 16-bit final address of the input/output variables

Output

**Var** (ANY_IN_UNITS_OF_WORDS)

Output variable for which the 16-bit final address is needed

## Remarks

- The 16-bit final address is yielded at input/output **AdrLast** based on the variable at input/output **Var**. This input or output has to be directly attached to the 16-bit input or output of a basic function.
- Only for LD and FBD Editors: Use “Input instruction” or “Output instruction” from the “Instructions” pane to insert the instruction required into the programming window.

## Example

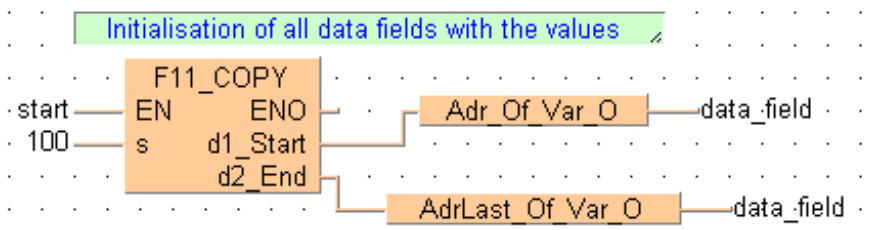
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activation of the function |
| 1 | VAR | data_field | ARRAY [0..5] OF INT | [6(111)] | result: [6(100)] |

### LD body

When the variable **start** is set to TRUE, the function is carried out. The function copies the value 100 into all elements of the data field, i.e. all six elements in **data_field** have the value 100.

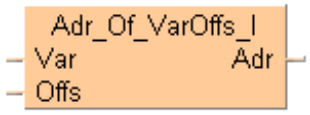


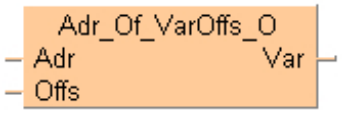

Further example see: Example (page 39)

# Adr_Of_VarOffs

Returns the input or output address with offset

This function returns the address of a variable with offset at the input or output of a basic function.




## Parameters

Input

**Adr** (ANY)

This pin must be connected to the input/output of a basic function for which the data type INT, WORD is allowed

Yields the 16-bit offset address of the input/output variable

**Offs** (INT)

Input/output variable for which the 16-bit offset address is needed

Output

**Var** (ANY_IN_UNITS_OF_WORDS)

Output variable for which the 16-bit final address is needed

## Remarks

- From the variable at input/output **Var**, the 16-bit address, which is raised by the value given at input/output **Offs**, is yielded at input/output **Adr**. This input or output has to be directly attached to the 16-bit input or output of a basic function. This function can only be used to work with applied data of a string, i.e. with characters without the string header.
- Only for LD and FBD Editors: Use “Input instruction” or “Output instruction” from the “Instructions” pane to insert the instruction required into the programming window.

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global variables

| | Class | Identifier | FP address | IEC address | Type | Initial |
|---|---|---|---|---|---|---|
| 1 | VAR_GLOBAL | ReceiveBuffer | DT0 | %MW5.0 | ARRAY [0..10] OF WORD | [11(0)] |

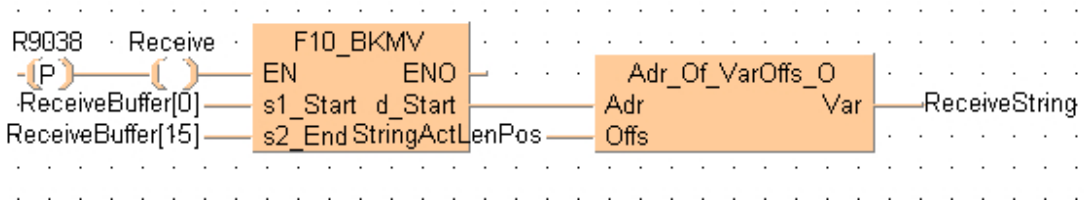
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | ReceiveBuffer | ARRAY [0..15] OF WORD | [16(0)] |
| 1 | VAR | Receive | BOOL | FALSE |
| 2 | VAR | ReceiveString | STRING[30] | '' |
| 3 | VAR_CONSTANT | StrigTotLenPos | INT | 0 |
| 4 | VAR_CONSTANT | StrigActLenPos | INT | 1 |
| 5 | VAR_CONSTANT | StrigHeaderSize | INT | 2 |

### LD body

When **Receive** is set and the reception-end character is received, i.e. R9038 is set, the characters in the buffer **ReceiveBuffer** received via the serial interface are copied to the applied data area of the string **ReceiveString**. The number of received characters is stored in **ReceiveBuffer[0]**; then follow the characters of the string received. Hence, **ReceiveBuffer** can be copied directly into this area of the string from this position.(see data type STRING). The position of the buffer **ReceiveBuffer** has to be assigned in the global variable list according to the settings in system regísters 417 and 418 (for detailed information, please refer to Setting receive buffer for CPU (page 501)).



# 2 Arithmetic instructions

# ABS

Absolute value

**ABS** calculates the value in the accumulator into an absolute value. The result is saved in the output variable.


## Parameters

Input

**Unnamed input** (INT, DINT, UINT, UDINT, REAL, LREAL)

Input data type

Output

**Unnamed output** (INT, DINT, UINT, UDINT, REAL, LREAL)

Output as input: absolute value

## Example

### POU header

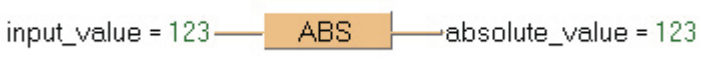
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | input_value | INT | -123 |
| 1 | VAR | absolute_value | INT | 0 |

This example uses variables. You can also use a constant for the input variable.

### LD body

**input_value** of the data type INTEGER is converted into an absolute value of the data type INTEGER. The converted value is written into **absolute_value**.



# ACOS

Arccosine

**ACOS** calculates the arccosine of the input variable and writes the angle data in radians into the output variable. The function returns a value from 0.0 to π.


## Parameters

Input

**Unnamed input** (REAL, LREAL)

Input value between -1 and +1

Output

**Unnamed output** (REAL, LREAL)

Output as input: arccosine of input value in radians

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if the input variable does not have the data type REAL, LREAL or if the input variable is not ≥-1.0 and ≤1.0

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if the input variable does not have the data type REAL, LREAL or if the input variable is not ≥-1.0 and ≤1.0

**sys_bIsEqual** (turns to TRUE and remains TRUE)

if the output variable is zero

**sys_bIsCarry** (turns to TRUE and remains TRUE)

if the processing result overflows the output variable

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | REAL | 0.0 | number between -1 and +1 |
| 1 | VAR | output_value | REAL | 0.0 | angle data in radians 0.0 to pi |

This example uses variables. You can also use a constant for the input variable.

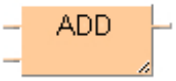
### LD body

The arc cosine of **input_value** is calculated and written into **output_value**.

# ADD

Add

This function adds the input variables **IN1** + **IN2** +... (up to 28 input contacts) and writes the addition result into the output variable.




## Parameters

Input

**Unnamed input** (INT, DINT, UINT, UDINT, REAL, LREAL)

1st input: augend

**Unnamed input** (INT, DINT, UINT, UDINT, REAL, LREAL)

2nd input: addend

Output

**Unnamed output** (INT, DINT, UINT, UDINT, REAL, LREAL)

Output as input: sum

## Remarks

- All operands must be of the same data type.
- This function can be expanded to a maximum of 28 input contacts, see also modifying elements.

## Example

### POU header

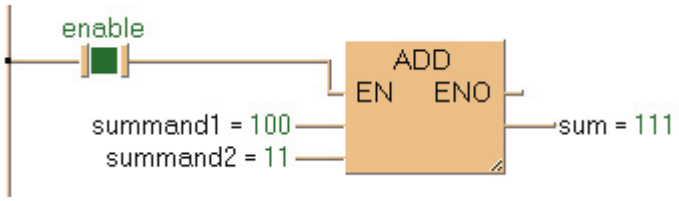
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | enable | BOOL | FALSE |
| 1 | VAR | summand_1 | INT | 0 |
| 2 | VAR | summand_2 | INT | 0 |
| 3 | VAR | sum | INT | 0 |

In this example the input variables (**summand_1**, **summand_2** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

## LD body

If **enable** is set (TRUE), **summand_1** is added to **summand_2**. The result is written into **sum**.



# ASIN

Arcsine

**ASIN** calculates the arcsine of the input variable and writes the angle data in radians into the output variable. The function returns a value from - π/2 to π/2.


## Parameters

Input

**Unnamed input** (REAL, LREAL)

Input value between -1 and +1

Output

**Unnamed output** (REAL, LREAL)

Output as input: arcsine of input value in radians

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if input variable does not have the data type REAL, LREAL or input variable is not ≥ -1.0 and ≤ 1.0

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if input variable does not have the data type REAL, LREAL or input variable is not ≥ -1.0 and ≤ 1.0

**sys_bIsEqual** (turns to TRUE and remains TRUE)

if output variable is zero

**sys_bIsCarry** (turns to TRUE for one scan)

if processing result overflows the output variable

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | REAL | 0.0 | number between -1 and +1 |
| 1 | VAR | output_value | REAL | 0.0 | angle data in radians -Pi/2 to Pi/2 |

This example uses variables. You can also use a constant for the input variable.

### LD body

The arc sine of **input_value** is calculated and written into **output_value**.

# ATAN

Arctangent

**ATAN** calculates the arctangent of the input variable (value ± 52707176) and writes the angle data in radians into the output variable. The function returns a value greater than -π/2 and smaller than π/2.


## Parameters

Input

**Unnamed input** (REAL, LREAL)

Input value between -52707176 and +52707176

Output

**Unnamed output** (REAL, LREAL)

Output as input: arctangent of input value in radians

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

if input variable does not have the data type REAL, LREAL or input variable ≥ 52707176

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

if input variable does not have the data type REAL, LREAL or input variable ≥ 52707176

**sys_blsEqual** (turns to TRUE and remains TRUE)

if output variable is zero

**sys_blsCarry** (turns to TRUE for one scan)

if processing result overflows the output variable

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | REAL | 0.0 | number between +/-52707176 |
| 1 | VAR | output_value | REAL | 0.0 | angle in radians >-Pi/2 and <Pi/2 |

This example uses variables. You can also use a constant for the input variable.

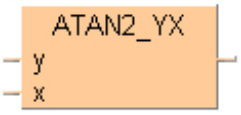
### LD body

The arc tangent of **input_value** is calculated and written into **output_value**.

# ATAN2_YX

Returns the angle φ of the Cartesian coordinates (x,y)

**ATAN2_YX** returns the angle φ of the Cartesian coordinates (x,y) within the range of -π to +π.




## Parameters

Input

**y** (REAL)

Cartesian y coordinate

**x** (REAL)

Cartesian x coordinate

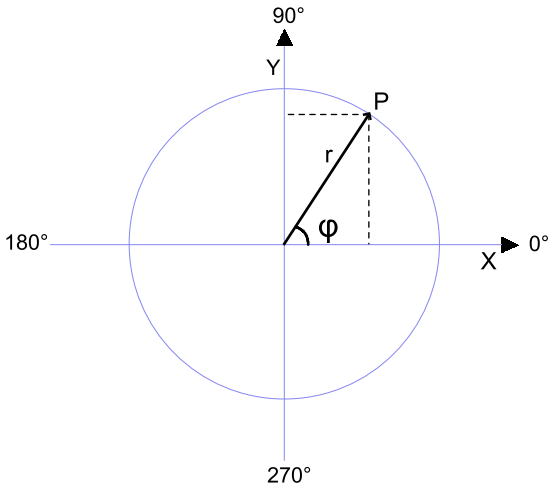
Output

**VAR_OUT** (REAL)

result in radians

## Remarks

Each position **P** of the two-dimensional coordinates can be defined by Cartesian coordinates P(x,y) or by polar coordinates P(r,φ) (r = radius, φ = angle).




Define **ATAN2_YX** as follows:

| **ATAN2_YX(y,x)** | **x** | **y** |
|---|---|---|
| $\arctan\frac{y}{x}$ | x > 0 | |
| $\arctan\frac{y}{x}+\pi$ | x < 0 | y≥0 |
| $\arctan\frac{y}{x}-\pi$ | | y < 0 |
| $+\frac{\pi}{2}$ | x = 0 | y > 0 |
| $-\frac{\pi}{2}$ | | y < 0 |
| 0 | | y = 0 |

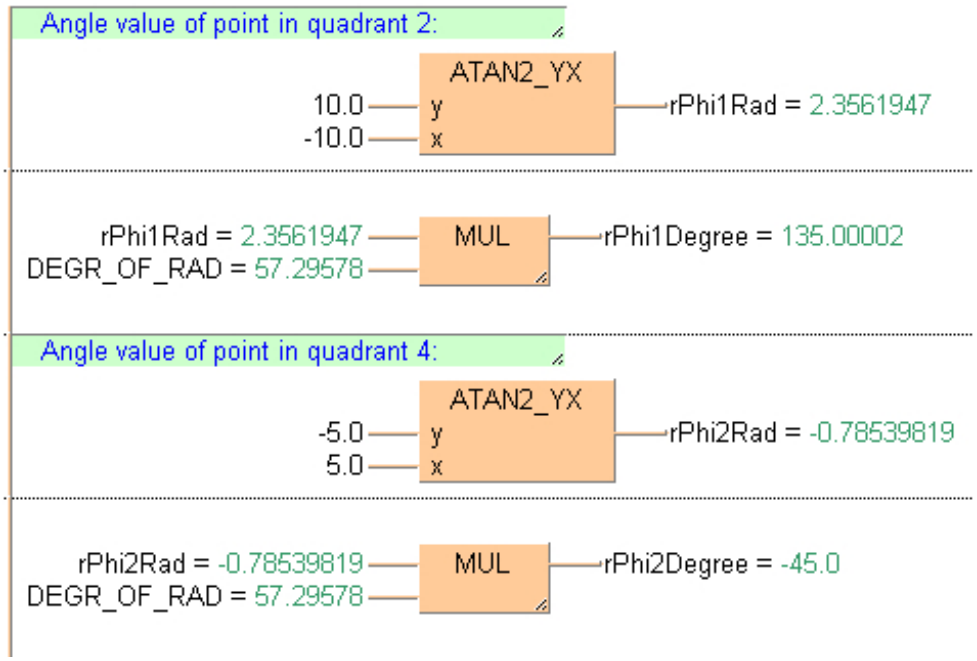
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | rPhi1Rad | REAL | 0.0 |
| 1 | VAR | rPhi2Rad | REAL | 0.0 |
| 2 | VAR | rPhi1Degree | REAL | 0.0 |
| 3 | VAR | rPhi2Degree | REAL | 0.0 |
| 4 | VAR_CONSTANT | DEGR_OF_RAD | REAL | 57.295779513082320876798154814105 |
| 5 | VAR | bCalculatePhi1 | BOOL | FALSE |

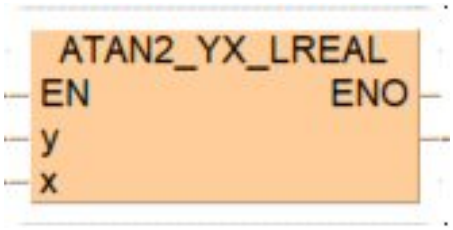
### LD body



# ATAN2_YX_LREAL

Returns the angle φ of the Cartesian coordinates (x,y) with LREAL arguments

**ATAN2_YX_LREAL** returns the angle φ of the Cartesian coordinates (x,y) within the range of -π to +π.




## Parameters

Input

**y** (LREAL)

Cartesian y coordinate

**x** (LREAL)

Cartesian x coordinate

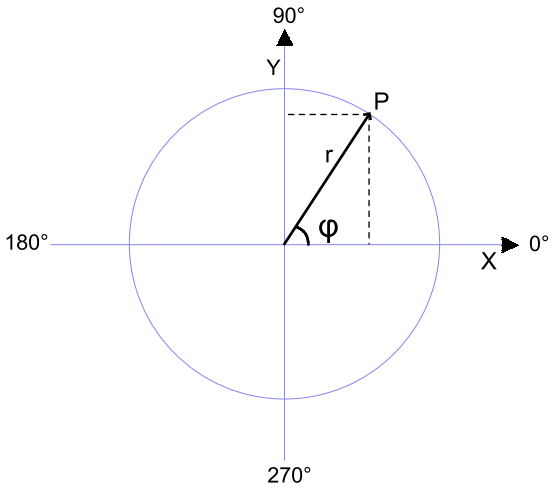
Output

**VAR_OUT** (LREAL)

result in radians

## Remarks

Each position **P** of the two-dimensional coordinates can be defined by Cartesian coordinates P(x,y) or by polar coordinates P(r,φ) (r = radius, φ = angle).




Define **ATAN2_YX** as follows:

| **ATAN2_YX(y,x)** | **x** | **y** |
|---|---|---|
| $\arctan\frac{y}{x}$ | x > 0 | |
| $\arctan\frac{y}{x}+\pi$ | x < 0 | y≥0 |
| $\arctan\frac{y}{x}-\pi$ | | y < 0 |
| $+\frac{\pi}{2}$ | x = 0 | y > 0 |
| $-\frac{\pi}{2}$ | | y < 0 |
| 0 | | y = 0 |

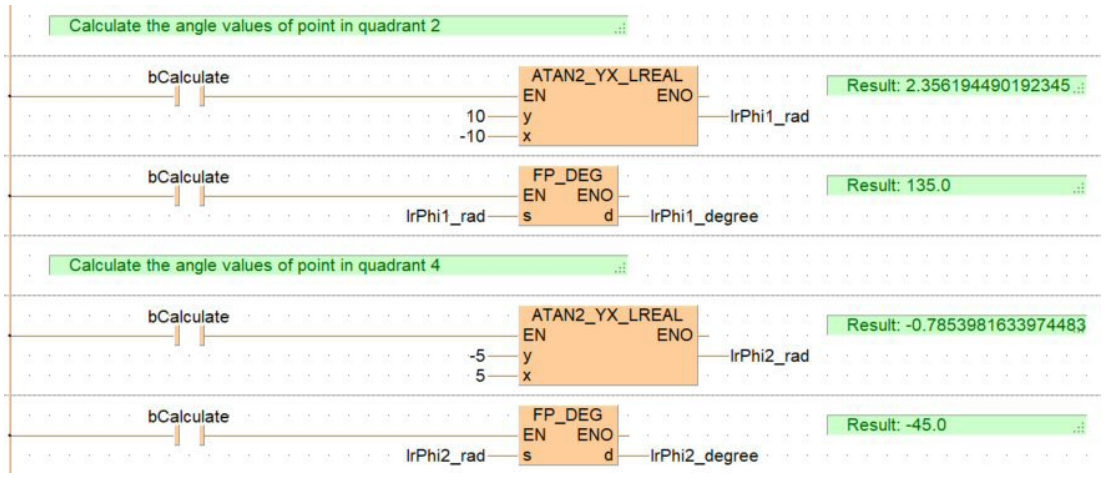
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | lrPhi1_rad | LREAL | 0 |
| 2 | VAR | lrPhi2_rad | LREAL | 0 |
| 3 | VAR | lrPhi1_degree | LREAL | 0 |
| 4 | VAR | lrPhi2_degree | LREAL | 0 |
| 5 | VAR | bCalculate | BOOL | FALSE |

### LD body



# COS

Cosine

**COS** calculates the cosine of the input variable and writes the result into the output variable. The angle data has to be specified in radians (value < 52707176).


## Parameters

Input

**Unnamed input** (REAL, LREAL)

Input value, angle data in radians

Output

**Unnamed output** (REAL, LREAL)

Output as input: cosine of input value

## Remarks

The accuracy of the calculation decreases as the angle data specified in the input variable increases. Therefore, we recommend to enter angle data in radians ≥ -2π and ≤ 2π.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if input variable does not have the data type REAL, LREAL or input variable ≥ 52707176

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if input variable does not have the data type REAL, LREAL or input variable ≥ 52707176

**sys_bIsEqual** (turns to TRUE and remains TRUE)

if output variable is zero

**sys_bIsCarry** (turns to TRUE for one scan)

if processing result overflows the output variable

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | REAL | 0.0 | angle data in radians |
| 1 | VAR | output_value | REAL | 0.0 | cosine |

This example uses variables. You can also use a constant for the input variable.

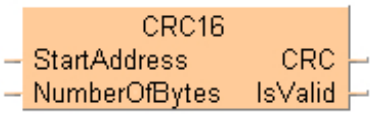
### LD body

The cosine of **input_value** is calculated and written into **output_value**.

# CRC16

Cyclic redundancy check

This function calculates the CRC16 (Cyclic Redundancy Check) for all PLC types by using **n** bytes (8 bits) specified with the parameter **NumberOfBytes** and the starting address **StartAddress**.




## Parameters

Input

**StartAddress** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Starting address for the calculation of the check sum. For PLCs which do not support the instruction **F70_BCC** with CRC16 calculation (FP0, FP5, FP10), the starting address must be in the DT or FL area.

**NumberOfBytes** (INT)

The number of bytes (8 bits), beginning with **StartAddress**, on which the CRC16 calculation is performed.

Output

**CRC** (WORD, INT, UINT)

The calculated check sum, which is only valid if the flag **IsValid** is set to TRUE.

**IsValid** (BOOL)

Flag indicating whether the calculated check sum is valid or not.
For PLCs which do not support the instruction **F70_BCC** with CRC16 calculation (FP0, FP5, FP10) the CRC is not valid:

- during the first eight execution scans when an internal table is built
- if the address area of the variable **StartAddress** is not in the DT or FL area.

For PLCs that support the instruction **F70_BCC** with CRC16 calculation, the CRC is always valid.

## Remarks

Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_CRC** (page 851)

Depending on the PLC type, one of the following two implementations of the function will be used:

- PLCs which support the instruction **F70_BCC** with the parameter s1=10 to calculate CRC16 (FP-e, FP-Sigma, FP2, FP2SH, FP10SH) use **F70_BCC** directly.
- For the other PLCs (FP0, FP0R, FP3, FP5, FP10), a sub-program making an explicit CRC16 calculation is called. The following restrictions apply to this sub-program:
  - During the first eight execution scans an internal table is built. During this time, no check sum is calculated, and the output **IsValid** remains FALSE. After that, the check sum is calculated, and the output **IsValid** is set to TRUE.
  - **StartAddress** requires an address in the DT or FL area.

Note

- The number of steps can increase up to approx. 200 when **CRC16** is used as a sub-program.
- When programming, please be aware that a certain amount of time is needed to build the internal table and to calculate the check sum, especially for large data volumes.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Array1 | ARRAY [0..10] OF INT | [0,1,2,3,4,5,6,7,8,9,10] |
| 1 | VAR | ARRAY1_BYTES | INT | 22 |
| 2 | VAR | Array1Crc | WORD | 0 |
| 3 | VAR | CrcIsvalid | BOOL | FALSE |

### LD body

# EXP

Exponent of input variable to base e

**EXP** calculates the power of the input variable to the base e (Euler's number = 2.7182818) and writes the result into the output variable. The input variable has to be greater than -87.33 and smaller than 88.72. This function is the reversion of the **LN** function.


## Parameters

Input

**Unnamed input** (REAL, LREAL)

Input value between -87.33 and +88.72

Output

**Unnamed output** (REAL, LREAL)

Output as input: exponent of input variable to base e

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if input variable does not have the data type REAL, LREAL
- if input variable is not > -87.33 and < 88.72

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if input variable does not have the data type REAL, LREAL
- if input variable is not > -87.33 and < 88.72

**sys_bIsEqual** (turns to TRUE and remains TRUE)

if output variable is zero

**sys_bIsCarry** (turns to TRUE for one scan)

if processing result overflows the output variable

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | REAL | 0.0 | > -87.33 and < 88.72 |
| 1 | VAR | output_value | REAL | 0.0 | number > 0 |

This example uses variables. You can also use a constant for the input variable.

### LD body

The power of **input_value** is calculated to the base e and written into **output_value**.

# EXPT

Raises 1st input variable by the power of the 2nd input variable

**EXPT** raises the first input variable to the power of the second input variable (OUT = $IN1^{IN2}$) and writes the result into the output variable.


## Parameters

Input

**Unnamed input** (REAL, LREAL)

Input value

**Unnamed input** (REAL, LREAL)

Exponent of the input value

Output

**Unnamed output** (REAL, LREAL)

Output as 1st input: result

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

if first and the second input variable do not have the data type REAL, LREAL

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

if first and the second input variable do not have the data type REAL, LREAL

**sys_blsEqual** (turns to TRUE and remains TRUE)

if output variable is zero

**sys_blsCarry** (turns to TRUE for one scan)

if processing result overflows the output variable

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value_1 | REAL | 0.0 | number from -1.70141x10^38 to |
| 1 | VAR | input_value_2 | REAL | 0.0 | number from -1.70141x10^38 to |
| 2 | VAR | output_value | REAL | 0.0 | number from -1.70141x10^38 to 1.70141x10^38 |
| 3 | VAR | | | | |

In this example the input variables **input_value_1** and **input_value_2** have been declared. Instead, you may enter constants directly at the input contacts of a function.

### LD body

**Input_value_1** is raised to the power of **input_value_2**. The result is written into **output_value**.

# DIV

Divide

**DIV** divides the value of the first input variable by the value of the second.


## Parameters

Input

**Unnamed input** (INT, DINT, UINT, UDINT, REAL, LREAL)

1st input: dividend

**Unnamed input** (INT, DINT, UINT, UDINT, REAL, LREAL)

2nd input: divisor

Output

**Unnamed output** (INT, DINT, UINT, UDINT, REAL, LREAL)

Output as input: result

## Remarks

Input and output variables must be of one of the noted data types. All operands must be of the same data type.

# Example

## POU header

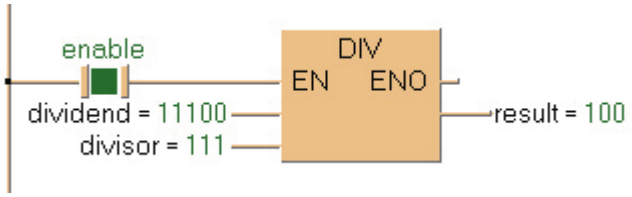
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | enable | BOOL | FALSE |
| 1 | VAR | dividend | INT | 0 |
| 2 | VAR | divisor | INT | 0 |
| 3 | VAR | result | INT | 0 |

In this example the input variables **dividend**, **divisor** and **enable** have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

### LD body

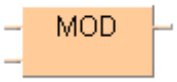
If **enable** is set (TRUE), **dividend** is divided by **divisor**. The result is written into **result**.



# MOD

Modular arithmetic division, remainder stored in output variable

**MOD** divides the value of the first input variable by the value of the second. The remainder of the integral division (e.g. 5 : 2 = 2, **remainder = 1**) is written into the output variable.




## Parameters

Input

**Unnamed input** (INT, DINT, UINT, UDINT)

1st input: dividend

**Unnamed input** (INT, DINT, UINT, UDINT)

2nd input: divisor

Output

**Unnamed output** (INT, DINT, UINT, UDINT)

Output as input: remainder

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | dividend | INT | 11 | |
| 1 | VAR | divisor | INT | 4 | |
| 2 | VAR | remainder | INT | 0 | 11 divided by 4 = 2 with remainder of 3<br>3 is written into output variable |
| 3 | VAR | | | | |

### LD body

This example uses variables. You may also use constants for the input variables. Dividend (11) is divided by divisor (4). The remainder (3) of the division is written in **remainder**.


dividend = 11
divisor = 4
MOD
remainder = 3

# MUL

Multiply

**MUL** multiplies the values of the input variables with each other and writes the result into the output variable.


## Parameters

Input

**Unnamed input** (INT, DINT, UINT, UDINT, REAL, LREAL)

1st input: multiplicand

**Unnamed input** (INT, DINT, UINT, UDINT, REAL, LREAL)

2nd input: multiplicator

Output

**Unnamed output** (INT, DINT, UINT, UDINT, REAL, LREAL)

Output as input: result

## Remarks

- All operands must be of the same data type.
- This function can be expanded to a maximum of 28 input contacts, see also modifying elements.

## Example

### POU header

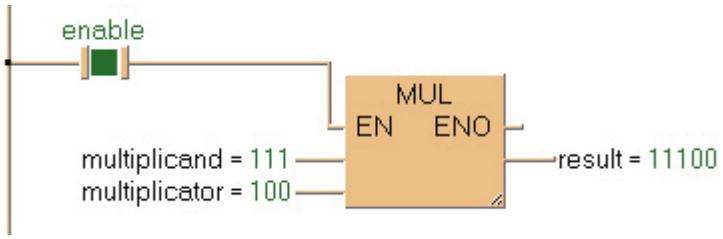
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | enable | BOOL | FALSE |
| 1 | VAR | multiplicand | INT | 0 |
| 2 | VAR | multiplicator | INT | 0 |
| 3 | VAR | result | INT | 0 |

In this example the input variables (**multiplicand**, **multiplicator** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

## LD body

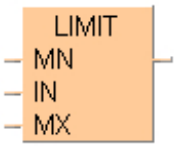
If **enable** is set (TRUE), the **multiplicant** is multiplied with the **multiplicator**. The result will be written into **result**.



# LIMIT

Limit value for input variable

In **LIMIT** the 1st input variable forms the lower and the 3rd input variable the upper limit value. If the 2nd input variable is within this limit, it will be transferred to the output variable. If it is above this limit, the upper limit value will be transferred; if it is below this limit the lower limit value will be transferred.




## Parameters

Input

**MN** (INT, DINT, UINT, UDINT, REAL, LREAL)

1st input: lower limit

**IN** (INT, DINT, UINT, UDINT, REAL, LREAL)

2nd input: value compared to upper and lower limit

**MX** (INT, DINT, UINT, UDINT, REAL, LREAL)

3rd input: upper limit

Output

**VAR_OUT** (INT, DINT, UINT, UDINT, REAL, LREAL)

Output as input: result, 2nd input value if between upper and lower limit, otherwise the upper or lower limit

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

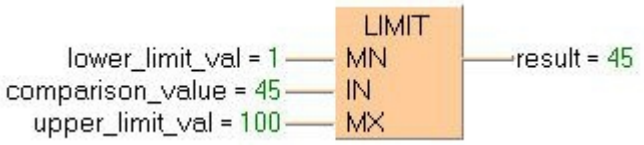
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | lower_limit_var | INT | 0 | all types allowed |
| 1 | VAR | comparision_value | INT | 0 | all types allowed |
| 2 | VAR | upper_limit_val | INT | 0 | all types allowed |
| 3 | VAR | result | INT | 0 | all types allowed |

In this example the input variables (**lower_limit_val**, **comparison_value** and **upper_limit_val**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

## LD body

**lower_limit_val** and **upper_limit_val** form the range where the **comparison_value** has to be, if it has to be transferred to **result**.

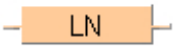
- If the **comparison_value** is above the **upper_limit_val**, the value of **upper_limit_val** will be transferred to **result**.
- If it is below the **lower_limit_val**, the value of **lower_limit_val** will be transferred to **result**.



# LN

Natural logarithm

**LN** calculates the logarithm of the input variable (value > 0.0) to the base e (Euler's number = 2.7182818) and writes the result into the output variable. This function is the reversion of the **EXP** function.




## Parameters

Input

**Unnamed input** (REAL, LREAL)

Input value

Output

**Unnamed output** (REAL, LREAL)

Output as input: natural logarithm of input value

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if input variable does not have the data type REAL, LREAL or input variable is not > 0.0

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if input variable does not have the data type REAL, LREAL or input variable is not > 0.0

**sys_bIsEqual** (turns to TRUE and remains TRUE)

if output variable is zero

**sys_bIsCarry** (turns to TRUE for one scan)

if processing result overflows the output variable

## Example

### POU header

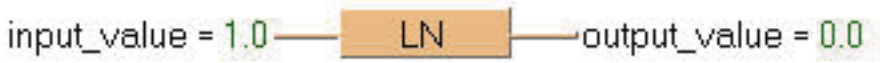
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | REAL | 0.0 | number > 0.0 |
| 1 | VAR | output_value | REAL | 0.0 | number unequal 0 |

This example uses variables. You can also use a constant for the input variable.

### LD body

The logarithm of **input_value** is calculated to the base e and written into **output_value**.



# LOG

Logarithm to the base 10

**LOG** calculates the logarithm of the input variable (value > 0.0) to the base 10 and writes the result into the output variable.


## Parameters

Input

**Unnamed input** (REAL, LREAL)

Input value

Output

**Unnamed output** (REAL, LREAL)

Output as input: logarithm of input value

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if input variable does not have the data type REAL, LREAL or input variable is not > 0.0

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if input variable does not have the data type REAL, LREAL or input variable is not > 0.0

**sys_bIsEqual** (turns to TRUE and remains TRUE)

if output variable is zero

**sys_bIsCarry** (turns to TRUE for one scan)

if processing result overflows the output variable

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | REAL | 0.0 | number > 0.0 |
| 1 | VAR | output_value | REAL | 0.0 | number unequal 0 |

This example uses variables. You can also use a constant for the input variable.

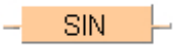
### LD body

The logarithm of **input_value** is calculated to the base 10 and written into **output_value**.

# SIN

Sine with radian input data

**SIN** calculates the sine of the input variable and writes the result into the output variable. The angle data has to be specified in radians (value < 52707176).




## Parameters

Input

**Unnamed input** (REAL, LREAL)

Input value, angle data in radians

Output

**Unnamed output** (REAL, LREAL)

Output as input: SINE of input value

## Remarks

The accuracy of the calculation decreases as the angle data specified in the input variable increases. Therefore, we recommend to enter angle data in radians ≥ -2π and ≤ 2π.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if input variable does not have the data type REAL, LREAL or input variable ≥ 52707176

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if input variable does not have the data type REAL, LREAL or input variable ≥ 52707176

**sys_bIsEqual** (turns to TRUE and remains TRUE)

if output variable is zero

**sys_bIsCarry** (turns to TRUE for one scan)

if processing result overflows the output variable

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | REAL | 0.0 | number >= 0 |
| 1 | VAR | output_value | REAL | 0.0 | number >= 0 |

This example uses variables. You can also use a constant for the input variable.

### LD body

The sine of **input_value** is calculated and written into **output_value**.

# SQRT

Square root

**SQRT** calculates the square root of an input variable of the data type REAL, LREAL (value ≥ 0.0). The result is written into the output variable.


## Parameters

Input

**Unnamed input** (REAL, LREAL)

Input value

Output

**Unnamed output** (REAL, LREAL)

Output as input: square root of input value

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

if input variable does not have the data type REAL, LREAL or input variable is not ≥ 0.0

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

if input variable does not have the data type REAL, LREAL or input variable is not ≥ 0.0

**sys_blsEqual** (turns to TRUE and remains TRUE)

if output variable is zero

**sys_blsCarry** (turns to TRUE for one scan)

if processing result overflows the output variable

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | REAL | 0.0 | number >= 0 |
| 1 | VAR | output_value | REAL | 0.0 | number >= 0 |

This example uses variables. You can also use a constant for the input variable.

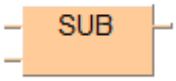
## LD body

The square root of **input_value** is calculated and written into **output_value**.

# SUB

Subtract

The content of the accumulator is subtracted from the operand defined in the operand field. The result is transferred to the accumulator.




## Parameters

Input

**Unnamed input** (INT, DINT, UINT, UDINT, REAL, LREAL)

1st input: minuend

**Unnamed input** (INT, DINT, UINT, UDINT, REAL, LREAL)

2nd input: subtrahend

Output

**Unnamed output** (INT, DINT, UINT, UDINT, REAL, LREAL)

Output as input: result

## Remarks

All operands must be of the same data type.

# Example

## POU header

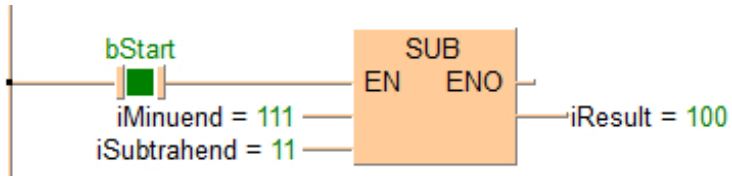
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | iMinuend | INT | 0 |
| 2 | VAR | iSubtrahend | INT | 0 |
| 3 | VAR | iResult | INT | 0 |

In this example the input variables (**iMinuend**, **iSubtrahend** and **bStart**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

## LD body

If **bStart** is set, **iSubtrahend** (data type INT) is subracted from **iMinuend**. The result will be written into **iResult** (data type INT).



# TAN

Tangent

**TAN** calculates the tangent of the input variable and writes the result into the output variable. The angle data has to be specified in radians (value < 52707176).


## Parameters

Input

**Unnamed input** (REAL, LREAL)

Input value in radians

Output

**Unnamed output** (REAL, LREAL)

Tangent of input value

## Remarks

The accuracy of the calculation decreases as the angle data specified in the input variable increases. Therefore, we recommend to enter angle data in radians ≥ -2π and ≤ 2π.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if input variable does not have the data type REAL, LREAL or input variable ≥ 52707176

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if input variable does not have the data type REAL, LREAL or input variable ≥ 52707176

**sys_bIsEqual** (turns to TRUE and remains TRUE)

if output variable is zero

**sys_bIsCarry** (turns to TRUE for one scan)

if processing result overflows the output variable

## Example

### POU header

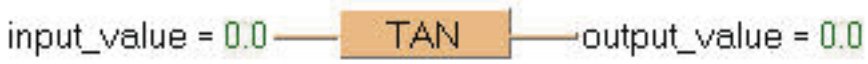
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | REAL | 0.0 | angle data in radians |
| 1 | VAR | output_value | REAL | 0.0 | tangent |

This example uses variables. You can also use a constant for the input variable.

### LD body

The tangent of **input_value** is calculated and written into **output_value**.



## 2.22 FP instructions

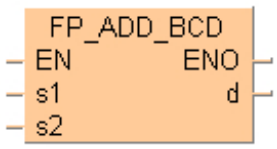
Tip

Advantages of FP instructions

# FP_ADD_BCD

BCD data addition

This FP instruction adds two BCD data specified by **s1** and **s2** if the trigger **EN** is TRUE. The result is stored in **d**.




## Parameters

Input

**s1**, **s2** (WORD, DWORD)

Addend

Output

**d** (WORD, DWORD)

Sum

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if data other than BCD data is specified.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if data other than BCD data is specified.

## Example

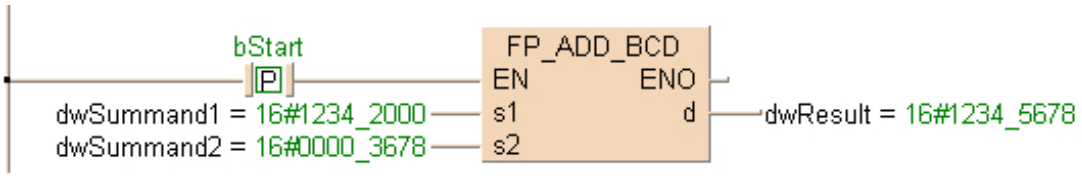
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | dwSummand1 | DWORD | 16#12342000 | summand at input s1 is added to summand at i... |
| 2 | VAR | dwSummand2 | DWORD | 16#00003678 | result after 0->1 leading edge from start: 16#12345678 |
| 3 | VAR | dwResult | DWORD | | |
| 4 | VAR | | | | |

## LD body

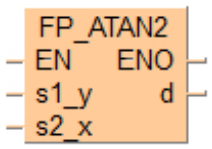
When the variable **bStart** changes from FALSE to TRUE, the function is carried out.



# FP_ATAN2

Convert coordinate data into radians

This FP instruction returns the angle φ of the Cartesian coordinates (x,y) within the range of -π to +π.




## Parameters

Input

**s1_y** (ANY_REAL)

Cartesian y coordinate

**s2_x** (ANY_REAL)

Cartesian x coordinate

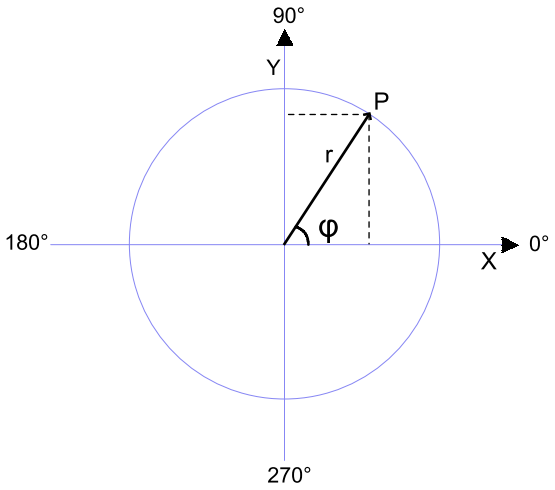
Output

**d** (REAL)

Result in radians

## Remarks

Each position **P** of the two-dimensional coordinates can be defined by Cartesian coordinates P(x,y) or by polar coordinates P(r,φ) (r = radius, φ = angle).




Define **FP_ATAN2** as follows:

| **ATAN2_YX(y,x)** | **x** | **y** |
|---|---|---|
| $\arctan\frac{y}{x}$ | x > 0 | |
| $\arctan\frac{y}{x}+\pi$ | x < 0 | y ≥ 0 |
| $\arctan\frac{y}{x}-\pi$ | | y < 0 |
| $+\frac{\pi}{2}$ | x = 0 | y > 0 |
| $-\frac{\pi}{2}$ | | y < 0 |
| Operation error | | y = 0 |

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if a non-real number is specified for **s1_y** (y coordinate) or **s2_x** (x coordinate)
- if 0.0 is specified for **s1_y** (y coordinate) and 0.0 for **s2_x** (x coordinate)

## Example

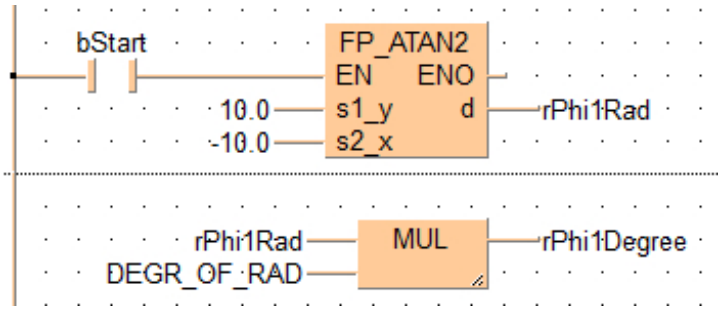
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |
| 1 | VAR | rPhi1Rad | REAL | 0.0 | |
| 2 | VAR | rPhi1Degree | REAL | 0.0 | |
| 3 | VAR_CONSTANT | DEGR_OF_RAD | REAL | 57.2957795130... | |

### LD body

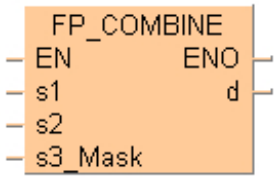
When the variable **bStart** is set to TRUE, the function is carried out.



# FP_COMBINE

Combine data

This FP instruction combines the two values at inputs **s1** and **s2** whereby the bit values at input **s3_Mask** determine the source of each bit. A bit value FALSE means the bit at input **s1** is used, a bit value TRUE means the bit at input **s2** is used. The result is stored in **d**.




## Parameters

Input

**s1**, **s2** (WORD, DWORD)

Values to be combined

**s3_Mask** (WORD, DWORD)

Input mask determining the source of each bit

Output

**d** (WORD, DWORD)

Combined result

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

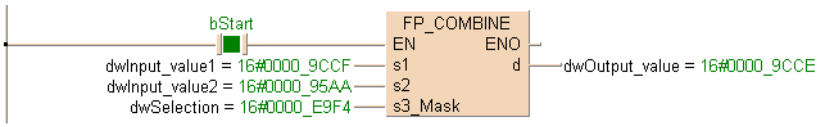
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | dwInput_value1 | DWORD | 2#00000000000000001001110011001111 | |
| 2 | VAR | dwInput_value2 | DWORD | 2#00000000000000001001010110101010 | |
| 3 | VAR | dwSelection | DWORD | 2#00000000000000001110100111110100 | |
| 4 | VAR | dwOutput_value | DWORD | 0 | the result after a 0->1 leading edge |

### LD body

When the variable **bStart** is set to TRUE, the function is carried out. The mask bits at **dwSelection** determine which bits are returned at the **dwOutput_value**. For mask bits which are TRUE (1), the corresponding bits of **dwInput_value1** are returned. For mask bits which are FALSE (0), the corresponding bits of **dwInput_value2** are returned.

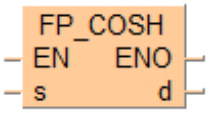



| -FP_COMBINE_LD | Structure |
|---|---|
| bStart | 2#1 at R250 |
| dwInput_value1 | 2#0000_0000_0000_0000_1001_1100_1100_1111 at DT1200 |
| dwInput_value2 | 2#0000_0000_0000_0000_1001_0101_1010_1010 at DT1202 |
| dwSelection | 2#0000_0000_0000_0000_1110_1001_1111_0100 at DT1204 |
| dwOutput_value | 2#0000_0000_0000_0000_1001_1100_1100_1110 at DT1206 |

# FP_COSH

Hyperbolic cosine

This FP instruction calculates the hyperbolic cosine of the angle data specified by input variable **s**. The result is stored in **d**.




## Parameters

Input

**s** (ANY_REAL)

Hyperbolic angle

Output

**d** (ANY_REAL)

Hyperbolic cosine of input value

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if a data type other than REAL is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if a data type other than REAL is specified.

## Example

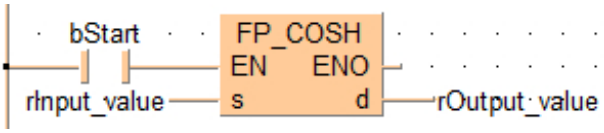
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | rInput_value | REAL | 0.0 |
| 2 | VAR | rOutput_value | REAL | 0.0 |

### LD body

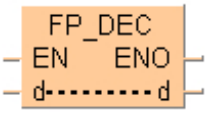
When the variable **start** is set to TRUE, the function is carried out.



# FP_DEC

Decrement

This FP instruction subtracts 1 from the data specified by **d** if the trigger **EN** is TRUE. The result is stored in **d**.




## Parameters

Input/output

**d** (INT, DINT, UINT, UDINT, REAL, LREAL)

Data to be decremented by 1 and result

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

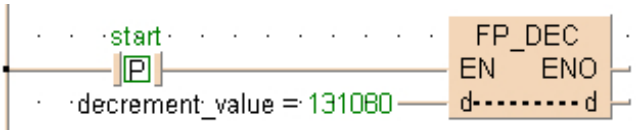
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | decrement_value | DINT | 131081 | result after a 0->1 leading |
| 2 | VAR | | | | edge from start: 131080 |

### LD body

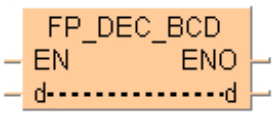
When the variable **bStart** is set to TRUE, the function is carried out.


start
P
decrement_value = 131080
FP_DEC
EN
ENO
d
d

# FP_DEC_BCD

BCD data decrement

This FP instruction subtracts 1 from the BCD data specified by **d** if the trigger **EN** is TRUE. The result is stored in **d**.




## Parameters

Input/output

**d** (WORD, DWORD)

Data to be decremented by 1 and result

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if data other than BCD data is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if data other than BCD data is specified.

## Example

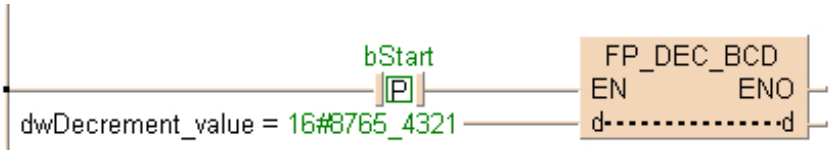
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | dwDecrement_value | DWORD | 16#87654322 | result after a 0->1 leding |

## LD body

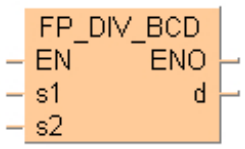
When the variable **bStart** changes from FALSE to TRUE, the function is carried out.


bStart
P
dwDecrement_value = 16#8765_4321
FP_DEC_BCD
EN
ENO
d
d

# FP_DIV_BCD

BCD data division

This FP instruction divides the BCD data specified by **s1** by **s2** if the trigger **EN** is TRUE. The result is stored in **d**.




## Parameters

Input

**s1** (WORD, DWORD)

Dividend

**s2** (WORD, DWORD)

Divisor

Output

**d** (WORD, DWORD)

Quotient

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if 0 is specified for **s2**.
- if data other than BCD data is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if 0 is specified for **s2**.
- if data other than BCD data is specified.

## Example

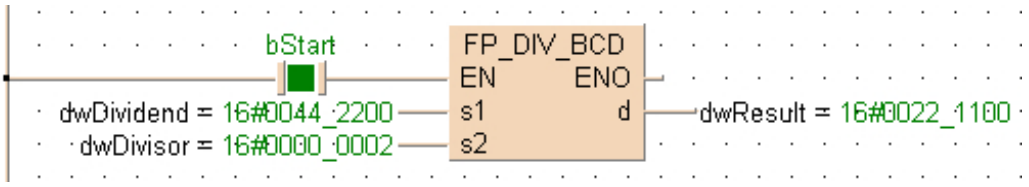
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the fuction |
| 1 | VAR | dwDividend | DWORD | 16#442200 | dividend |
| 2 | VAR | dwDivisor | DWORD | 16#2 | divisor |
| 3 | VAR | dwResult | DWORD | 0 | result after a 0->1 leading |

### LD body

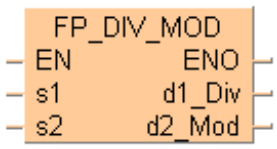
When the variable **bStart** is set to TRUE, the function is carried out.



# FP_DIV_MOD

Data division with remainder

This FP instruction divides the data specified by **s1** by **s2** if the trigger **EN** is TRUE. The integer quotient is stored in **d1_Div** and the remainder in **d2_Mod**.




## Parameters

Input

**s1** (INT, DINT, UINT, UDINT)

Dividend

**s2** (INT, DINT, UINT, UDINT)

Divisor

Output

**d1_Div** (INT, DINT, UINT, UDINT)

Integer quotient

**d2_Mod** (INT, DINT, UINT, UDINT)

Remainder

## Remarks

You can only use the data types UINT and UDINT with FP7 PLCs.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if 0 is specified for **s2**.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if 0 is specified for **s2**.

## Example

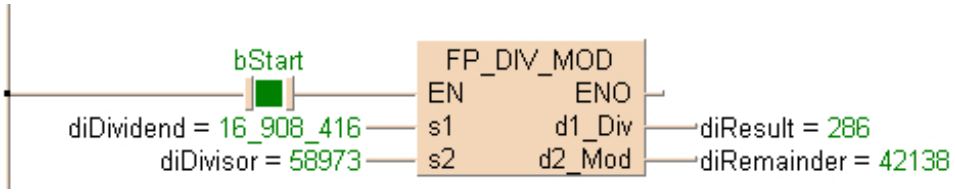
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the fuction |
| 1 | VAR | diDividend | DINT | 16908416 | dividend |
| 2 | VAR | diDivisor | DINT | 58973 | divisor |
| 3 | VAR | diResult | DINT | 0 | result after a 0->1 leading |
| 4 | VAR | diRemainder | DINT | 0 | remainder |
| 5 | VAR | | | | |

### LD body

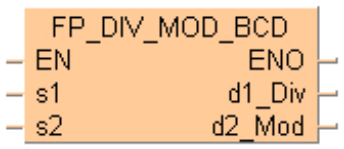
When the variable **bStart** is set to TRUE, the function is carried out.



# FP_DIV_MOD_BCD

BCD data division with remainder

This FP instruction divides the BCD data specified by **s1** by **s2** if the trigger **EN** is TRUE. The integer quotient is stored in **d1_Div** and the remainder in **d2_Mod**.




## Parameters

Input

**s1** (WORD, DWORD)

Dividend

**s2** (WORD, DWORD)

Divisor

Output

**d1_Div** (WORD, DWORD)

Integer quotient

**d2_Mod** (WORD, DWORD)

Remainder

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if 0 is specified for **s2**.
- if data other than BCD data is specified.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if 0 is specified for **s2**.
- if data other than BCD data is specified.

## Example

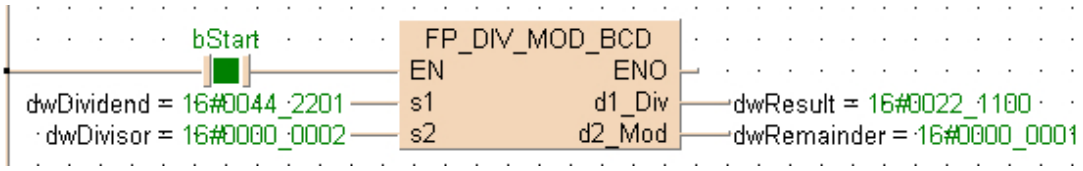
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the fuction |
| 1 | VAR | dwDividend | DWORD | 16#442201 | dividend |
| 2 | VAR | dwDivisor | DWORD | 16#2 | divisor |
| 3 | VAR | dwResult | DWORD | 0 | result after a 0->1 leading |
| 4 | VAR | dwRemainder | DWORD | 0 | remainder: 16#1 |

### LD body

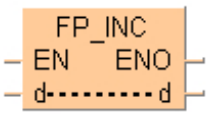
When the variable **bStart** is set to TRUE, the function is carried out.



# FP_INC

Increment

This FP instruction adds 1 to the data specified by **d** if the trigger **EN** is TRUE. The result is stored in **d**.




## Parameters

Input/output

**d** (INT, DINT, UINT, UDINT, REAL, LREAL)

Data to be incremented by 1 and result

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

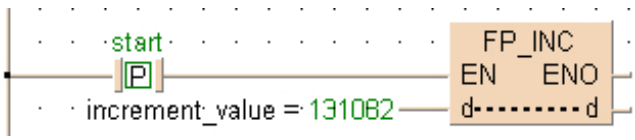
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | increment_value | DINT | 131081 | result after a 0->1 leading edge from start: 131082 |
| 2 | VAR | | | | |

### LD body

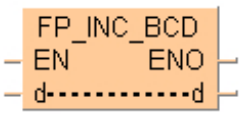
When the variable **bStart** changes from FALSE to TRUE, the function is carried out.


start
P
increment_value = 131082
FP_INC
EN
ENO
d
d

# FP_INC_BCD

BCD data increment

This FP instruction adds 1 to the BCD data specified by **d** if the trigger **EN** is TRUE. The result is stored in **d**.




## Parameters

Input/output

**d** (WORD, DWORD)

Data to be incremented by 1 and result

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if data other than BCD data is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if data other than BCD data is specified.

## Example

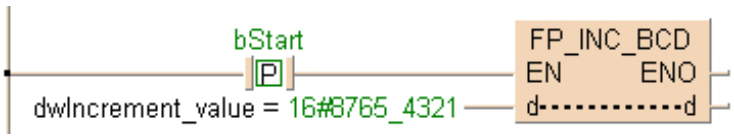
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | dwIncrement_value | DWORD | 16#87654320 | result after a 0->1 leding |

## LD body

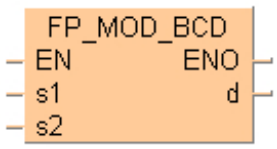
When the variable **bStart** changes from FALSE to TRUE, the function is carried out.


bStart
P
dwIncrement_value = 16#8765_4321
FP_INC_BCD
EN
ENO
d
d

# FP_MOD_BCD

Remainder of BCD data division

This FP instruction calculates the remainder of a BCD data division where the value specified by **s1** is divided by **s2**. The remainder is stored in **d**.




## Parameters

Input

**s1**, **s2** (WORD, DWORD)

Dividend, divisor

Output

**d** (WORD, DWORD)

Remainder

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if 0 is specified for **s2**.
- if data other than BCD data is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if 0 is specified for **s2**.
- if data other than BCD data is specified.

## Example

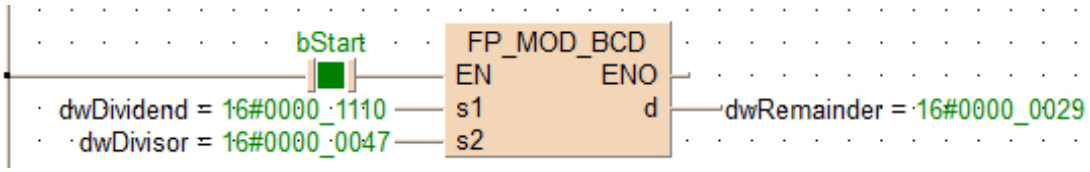
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the fuction |
| 1 | VAR | dwDividend | DWORD | 16#00001110 | dividend |
| 2 | VAR | dwDivisor | DWORD | 16#00000047 | divisor |
| 3 | VAR | dwRemainder | DWORD | 0 | result after 0->1 leading edge |

### LD body

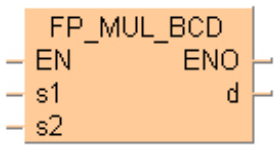
When the variable **bStart** is set to TRUE, the function is carried out.



# FP_MUL_BCD

BCD data multiplication

This FP instruction multiplies the BCD data specified by **s1** and **s2** if the trigger **EN** is TRUE. The result is stored in **d**.




## Parameters

Input

**s1** (WORD, DWORD)

Multiplicand

**s2** (WORD, DWORD)

Multiplier

Output

**d** (WORD, DWORD)

Product

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if data other than BCD data is specified.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if data other than BCD data is specified.

## Example

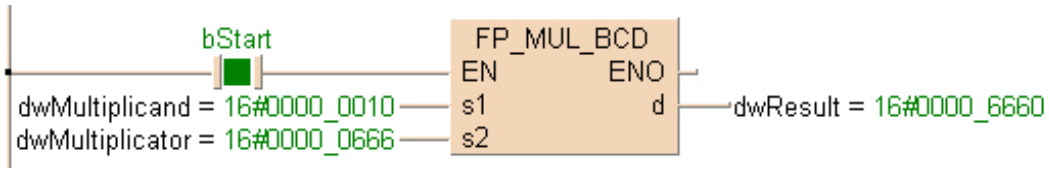
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | dwMultiplicand | DWORD | 16#00000010 | multiplicant |
| 2 | VAR | dwMultiplicator | DWORD | 16#00000666 | multiplicator |
| 3 | VAR | dwResult | DWORD | 0 | result after a 0->1 leading |

### LD body

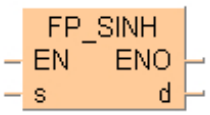
When the variable **bStart** is set to TRUE, the function is carried out.



# FP_SINH

Hyperbolic sine

This FP instruction calculates the hyperbolic sine of the angle data specified by input variable **s**. The result is stored in **d**.




## Parameters

Input

**s** (ANY_REAL)

Hyperbolic angle

Output

**d** (ANY_REAL)

Hyperbolic sine of input value

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if a data type other than REAL is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if a data type other than REAL is specified.

## Example

## POU header

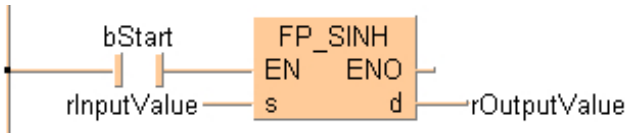
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |
| 1 | VAR | rInputValue | REAL | 0.0 | angle data in radians |
| 2 | VAR | rOutputValue | REAL | 0.0 | result |

This example uses variables. You can also use a constant for the input variable.

## LD body

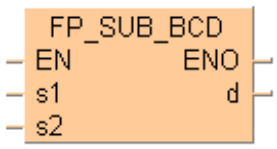
When the variable **bStart** is set to TRUE, the function is carried out. The sine of **rInputValue** is calculated and written into **rOutputValue**.



# FP_SUB_BCD

BCD data subtraction

This FP instruction subtracts the BCD data specified by **s2** from **s1** if the trigger **EN** is TRUE. The result is stored in **d**.




## Parameters

Input

**s1**, **s2** (WORD, DWORD)

Minuend, subtrahend

Output

**d** (WORD, DWORD)

Difference

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if data other than BCD data is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if data other than BCD data is specified.

## Example

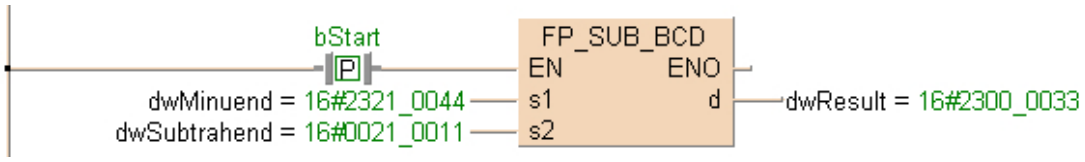
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | dwMinuend | DWORD | 16#23210044 | |
| 2 | VAR | dwSubtrahend | DWORD | 16#00210011 | this value will be subtracted |
| 3 | VAR | dwResult | DWORD | | result after 0->1 leading |

## LD body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out.



# FP_TANH

Hyperbolic tangent

This FP instruction calculates the hyperbolic tangent of the input variable. The result is stored in **d**.


## Parameters

Input

**s** (ANY_REAL)

Angle data in radians

Output

**d** (ANY_REAL)

Hyperbolic tangent of input value

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if a data type other than REAL is specified.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if a data type other than REAL is specified.

## Example

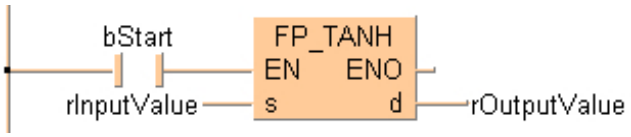
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |
| 1 | VAR | rInputValue | REAL | 0.0 | angle data in radians |
| 2 | VAR | rOutputValue | REAL | 0.0 | result |

## LD body

When the variable **start** is set to TRUE, the function is carried out. The tangent of **rInputValue** is calculated and written into **rOutputValue**.



## 2.23 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

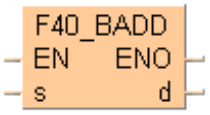
Advantages of IEC instructions

Advantages of FP instructions

# F40_BADD

4-digit BCD addition

The 4-digit BCD equivalent constant or 16-bit area for 4-digit BCD data specified by **s** and the 16-bit area for 4-digit BCD data specified by **d** are added together if the trigger **EN** is in the ON-state. The result is stored in **d**.




## Parameters

Input

**s** (WORD)

Addend, 16-bit area for 4-digit BCD data or equivalent constant

Output

**d** (WORD)

Augend and result, 16-bit area for 4-digit BCD data

### Example

①

| Bit | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|
| d | 0 0 1 0 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 |
| 16# (BCD) | 2 | 1 | 1 | 1 |

+

②

| Bit | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|
| s | 0 0 0 0 | 0 0 0 0 | 0 0 0 1 | 0 0 0 1 |
| 16# (BCD) | 0 | 0 | 1 | 1 |

↓

③

| Bit | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|
| d | 0 0 1 0 | 0 0 0 1 | 0 0 1 0 | 0 0 1 0 |
| 16# (BCD) | 2 | 1 | 2 | 2 |

(1) Example value 16#2111 (BCD)
(2) Example value 16#0011 (BCD)
(3) Result value 16#2122 (BCD) if trigger is ON

## Remarks

Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_ADD_BCD**

Note

When this instruction is used, the area for the augend **d** is overwritten by the added result. If you want to avoid the overwrite, we recommend using the instruction **F41_DBADD**.

## Error flags

**sys_bIsEqual** (turns to TRUE for one scan)

if the calculated result is 0.

**sys_bIsCarry** (turns to TRUE for one scan)

if the result exceeds the range of 4-digit

## Example

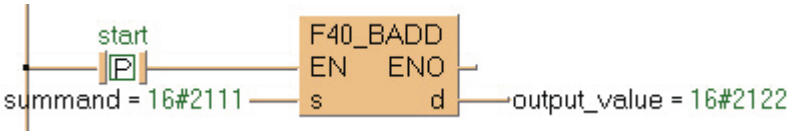
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | summand | WORD | 16#2111 | this value will be added |
| 2 | VAR | output_value | WORD | 16#0011 | result after 0->1 leading edge from start: 16#2122 |
| 3 | VAR | | | | |

## LD body

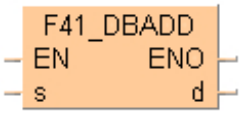
When the variable **start** changes from FALSE to TRUE, the function is carried out.



# F41_DBADD

8-digit BCD addition

The 8-digit BCD equivalent constant or 8-digit BCD data specified by **s** and the 8-digit BCD data specified by **d** are added together if the trigger **EN** is in the ON-state. The result is stored in **d**.




## Parameters

Input

**s** (DWORD)

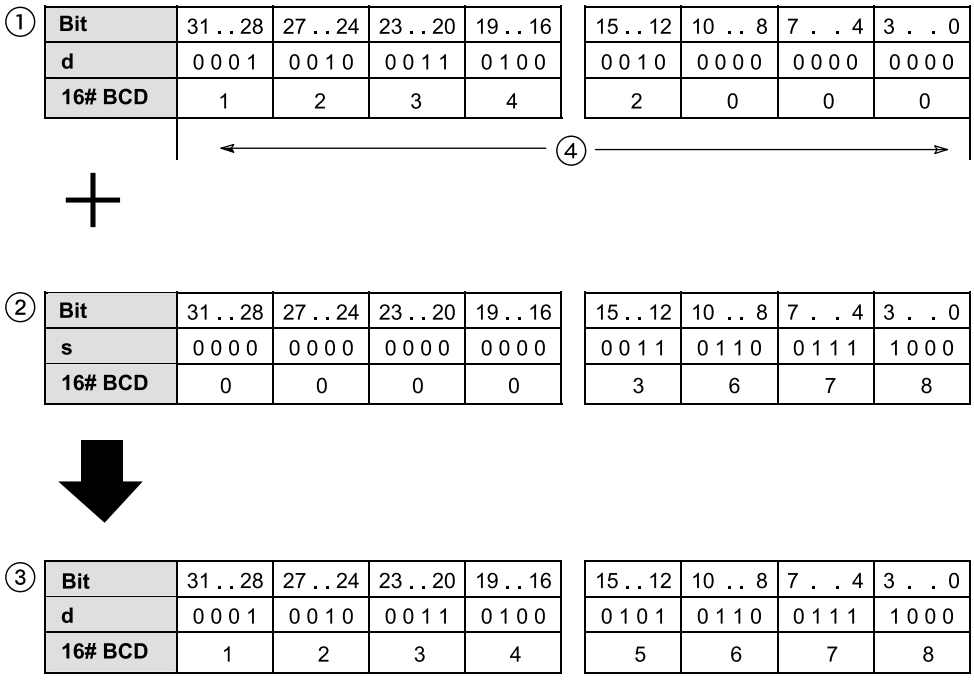
Addend, 32-bit area for 8-digit BCD data or equivalent constant

Output

**d** (DWORD)

Augend and result, 32-bit area for 8-digit BCD data

### Example



① 

| Bit | 31 . . 28 | 27 . . 24 | 23 . . 20 | 19 . . 16 | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|---|---|---|---|
| d | 0 0 0 1 | 0 0 1 0 | 0 0 1 1 | 0 1 0 0 | 0 0 1 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 |
| 16# BCD | 1 | 2 | 3 | 4 | 2 | 0 | 0 | 0 |

④

②

| Bit | 31 . . 28 | 27 . . 24 | 23 . . 20 | 19 . . 16 | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|---|---|---|---|
| s | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 1 1 | 0 1 1 0 | 0 1 1 1 | 1 0 0 0 |
| 16# BCD | 0 | 0 | 0 | 0 | 3 | 6 | 7 | 8 |

③

| Bit | 31 . . 28 | 27 . . 24 | 23 . . 20 | 19 . . 16 | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|---|---|---|---|
| d | 0 0 0 1 | 0 0 1 0 | 0 0 1 1 | 0 1 0 0 | 0 1 0 1 | 0 1 1 0 | 0 1 1 1 | 1 0 0 0 |
| 16# BCD | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

(1) Example value 16#12342000 (BCD)
(2) Example value 16#00003678 (BCD)
(3) Result value 16#12345678 (BCD) if trigger is ON
(4) 32-bit area

## Remarks

Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_ADD_BCD**

Note

When this instruction is used, the area for the augend **d** is overwritten by the added result. If you want to avoid the overwrite, we recommend using the instruction **F43_DBADD2**.

## Error flags

**sys_bIsEqual** (turns to TRUE for one scan)

if the calculated result is 0.

**sys_bIsCarry** (turns to TRUE for one scan)

if the result exceeds the range of 8-digit BCD data (overflow).

## Example

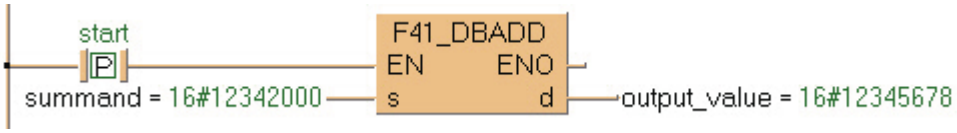
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | summand | DWORD | 16#12342000 | this value will be added |
| 2 | VAR | output_value | DWORD | 16#00003678 | result after 0->1 leading edge from start: 16#12345678 |
| 3 | VAR | | | | |

### LD body

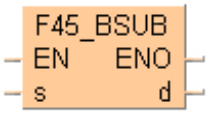
When the variable **start** changes from FALSE to TRUE, the function is carried out.



# F45_BSUB

4-digit BCD subtraction

Subtracts the 4-digit BCD equivalent constant or 16-bit area for 4-digit BCD data specified by **s** from the 16-bit area for 4-digit BCD data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.




## Parameters

Input

**s** (WORD)

Subtrahend, 16-bit area for 4-digit BCD data or equivalent constant

Output

**d** (WORD)

Minuend and result, 16-bit area for 4-digit BCD data

### Example

(1)

| Bit | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|
| d | 0 0 1 0 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 |
| 16# (BCD) | 2 | 1 | 1 | 1 |

−

(2)

| Bit | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|
| s | 0 0 0 0 | 0 0 0 0 | 0 0 0 1 | 0 1 0 1 |
| 16# (BCD) | 0 | 0 | 1 | 1 |

(4)

(3)

| Bit | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|
| d | 0 0 1 0 | 0 0 0 1 | 0 0 0 0 | 0 0 0 0 |
| 16# (BCD) | 2 | 1 | 0 | 0 |

(1) Example value 16#2111 (BCD)
(2) Example value 16#0011 (BCD)
(3) Result value 16#2100 (BCD)
(4) Trigger: ON

## Remarks

Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_SUB_BCD**

## Error flags

**sys_bIsEqual** (turns to TRUE for one scan)

if the calculated result is 0.

**sys_bIsCarry** (turns to TRUE for one scan)

if the result exceeds the range of 4-digit BCD data (overflow).

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | subtrahend | WORD | 16#0011 | this value will be subtracted |
| 2 | VAR | output_value | WORD | 16#2111 | result after 0->1 leading edge from start: 16#2100 |
| 3 | VAR | | | | |

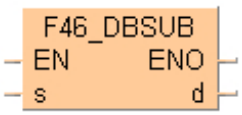
## LD body

When the variable **start** changes from FALSE to TRUE, the function is carried out.

# F46_DBSUB

8-digit BCD subtraction

Subtracts the 8-digit BCD equivalent constant or 8-digit BCD data specified by **s** from the 8-digit BCD data specified by **d** if the trigger **EN** is in the ON-state. The result is stored in **d**.




## Parameters

Input

**s** (DWORD)

Subtrahend, 32-bit area for 8-digit BCD data or equivalent constant

Output

**d** (DWORD)

Minuend and result, 32-bit area for 8-digit BCD data

### Example

(1)

| Bit | 31 . . 28 | 27 . . 24 | 23 . . 20 | 19 . . 16 | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|---|---|---|---|
| d | 0 0 1 0 | 0 0 1 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 0 | 0 0 0 0 | 0 1 0 0 | 0 1 0 0 |
| 16# BCD | 2 | 3 | 2 | 1 | 0 | 0 | 4 | 4 |

(4)

(2)

| Bit | 31 . . 28 | 27 . . 24 | 23 . . 20 | 19 . . 16 | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|---|---|---|---|
| s | 0 0 0 0 | 0 0 0 0 | 0 0 1 0 | 0 0 0 1 | 0 0 0 0 | 0 0 0 0 | 0 0 0 1 | 0 0 0 1 |
| 16# BCD | 0 | 0 | 2 | 1 | 0 | 0 | 1 | 1 |

(5)

(3)

| Bit | 31 . . 28 | 27 . . 24 | 23 . . 20 | 19 . . 16 | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|---|---|---|---|
| d | 0 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 1 1 | 0 0 1 1 |
| 16# BCD | 2 | 3 | 0 | 0 | 0 | 0 | 3 | 3 |

(1) Example value: 16#23210044 (BCD)
(2) Example value: 16#00210011 (BCD)
(3) Target value: 16#23000033 (BCD)
(4) 32-bit area
(5) Trigger: ON

## Remarks

Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_SUB_BCD**

## Error flags

**sys_bIsEqual** (turns to TRUE for one scan)

if the calculated result is 0.

**sys_bIsCarry** (turns to TRUE for one scan)

if the result exceeds the range of 8-digit BCD data (overflow).

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | subtrahend | DWORD | 16#00210011 | this value will be subtracted |
| 2 | VAR | output_value | DWORD | 16#23210044 | result after 0->1 leading edge from start: 16#23000033 |
| 3 | VAR | | | | |

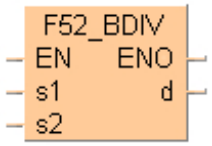
### LD body

When the variable **start** changes from FALSE to TRUE, the function is carried out.

# F52_BDIV

4-digit BCD division, destination can be specified

The 4-digit BCD equivalent constant or the 16-bit area for 4-digit BCD data specified by **s1** is divided by the 4-digit BCD equivalent constant or the 16-bit area for 4-digit BCD data specified by **s2** if the trigger **EN** is in the ON-state.




## Parameters

Input

**s1** (WORD)

Dividend, 16-bit area for BCD data or 4-digit BCD equivalent constant

**s2** (WORD)

Divisor, 16-bit area for BCD data or 4-digit BCD equivalent constant

Output

**d** (WORD)

Quotient, 16-bit area for BCD data (remainder stored in the system variable **sys_iDivRemainder**

### Example

(1)

| Bit | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|
| d | 0 0 0 0 | 0 0 0 0 | 0 0 1 1 | 0 1 1 1 |
| 16# (BCD) | 0 | 0 | 3 | 7 |

(2)

| Bit | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|
| s | 0 0 0 0 | 0 0 0 0 | 0 0 0 1 | 0 1 0 1 |
| 16# (BCD) | 0 | 0 | 1 | 5 |

 (5)

(3)

| Bit | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|
| d | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 1 0 |
| 16# (BCD) | 0 | 0 | 0 | 2 |

(4)

| Bit | 15 . . 12 | 10 . . 8 | 7 . . 4 | 3 . . 0 |
|---|---|---|---|---|
| DT9015 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 1 1 1 |
| 16# (BCD) | 0 | 0 | 0 | 7 |

(1) Example value 16#0037 (BCD)
(2) Example value 16#0015 (BCD)
(3) Result value 16#0002
(4) Remainder 16#0007
(5) Trigger: ON

## Remarks

The quotient is stored in the area specified by **d** and the remainder is stored in the system variable **sys_iDivRemainder**.

## Error flags

**sys_bIsEqual** (turns to TRUE for one scan)

if the calculated result is 0.

## Example

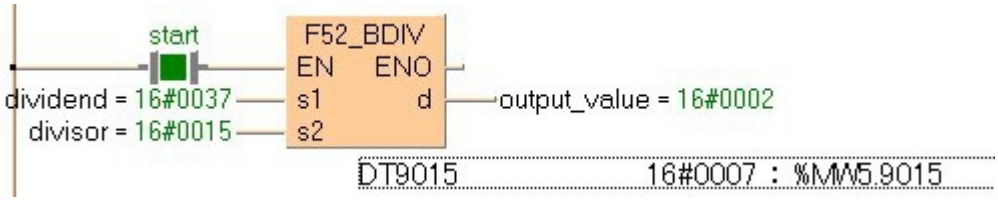
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | dividend | WORD | 16#0037 | dividend |
| 2 | VAR | divisor | WORD | 16#0015 | divisor |
| 3 | VAR | output_value | WORD | 0 | result after 0->1 leading edge |
| 4 | VAR | | | | from start: 16#0002 |

## LD body

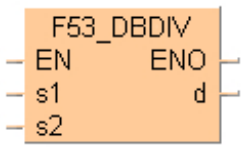
When the variable **start** is set to TRUE, the function is carried out.



# F53_DBDIV

8-digit BCD division, destination can be specified

The result is stored in the area specified by **d**, and the remainder is stored in the system variable **sys_diDDivRemainder**.




## Parameters

Input

**s1** (DWORD)

Dividend, 32-bit area for BCD data or 8-digit BCD equivalent constant

**s2** (DWORD)

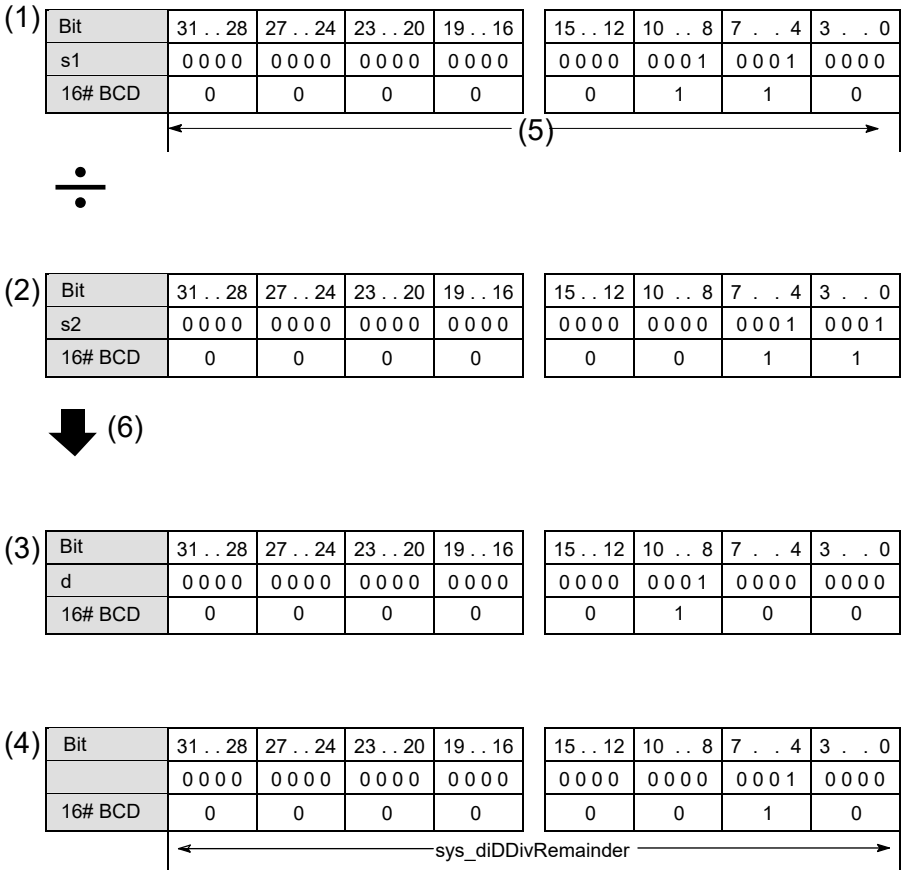
Divisor, 32-bit area for BCD data or 8-digit BCD equivalent constant

Output

**d** (DWORD)

Quotient, 32-bit area for BCD data (remainder stored in the system variable **sys_diDDivRemainder**

### Example




(1) Example value 16#00001110 (BCD)
(2) Example value 16#0000011 (BCD)
(3) Result value 16#00000100 (BCD) if trigger is ON
(4) Remainder 16#00000010 (BCD) if trigger is ON stored in **sys_diDDivRemainder**
(5) 32-bit area
(6) Trigger: ON

## Remarks

Instead of using this F instruction, we recommend using the corresponding FP7 instruction:

FP_MOD_BCD,FP_DIV_MOD_BCD,FP_DIV_BCD

## Error flags

**sys_bIsEqual** (turns to TRUE for one scan)

if the calculated result is 0.

## Example

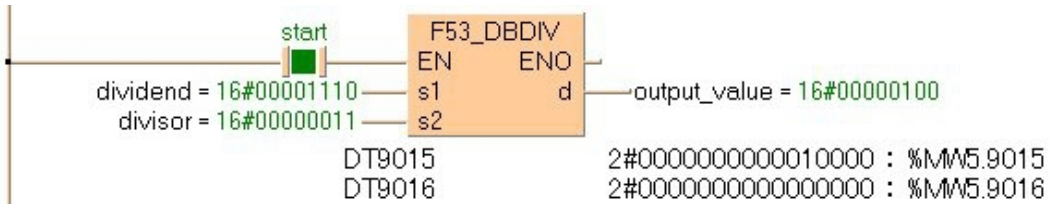
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | dividend | DWORD | 16#00001110 | dividend |
| 2 | VAR | divisor | DWORD | 16#00000011 | divisor |
| 3 | VAR | output_value | DWORD | 0 | result after 0->1 leading edge from start: 16#00000100 |
| 4 | VAR | | | | |

### LD body

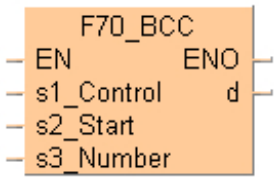
When the variable **start** is set to TRUE, the function is carried out.



# F70_BCC

Block check code calculation

Calculates the Block Check Character (**BCC**), which is used to detect errors in message transmission, of **s3_Number** bytes of ASCII data starting from the 16-bit area specified by **s2_Start** according to the calculation method specified by **s1_Control**. The Block Check Code (**BCC**) is stored in the lower byte of the 16-bit area specified by **d**. (BCC is one byte. The higher byte of **d** does not change.)




## Parameters

Input

**s1_Control** (INT)

Specifies BCC calculation method:

- 0: Addition (**SYS_BCC_CALCULATION_METHOD_ADD**)
- 1: Subtraction (**SYS_BCC_CALCULATION_METHOD_SUB**)
- 2: Exclusive OR (**SYS_BCC_CALCULATION_METHOD_XOR**)
- 16#A: CRC-16 (**SYS_BCC_CALCULATION_METHOD_CRC16**)

**s2_Start** (WORD, INT, UINT)

Starting 16-bit area to calculate BCC

**s3_Number** (INT)

Specifies number of bytes for BCC calculation

Output

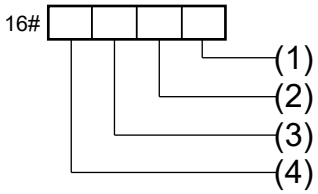
**d** (WORD, INT, UINT)

16-bit area for storing BCC

## Remarks

Instead of using this F instruction, we recommend using the corresponding FP7 instruction: FP_CRC, FP_BCC

Specifying the control code **s1_Control**




(1) Calculation method
0: Addition (**SYS_BCC_CALCULATION_METHOD_ADD**)
1: Subtraction (**SYS_BCC_CALCULATION_METHOD_SUB**)
2: Exclusive OR (**SYS_BCC_CALCULATION_METHOD_XOR**)
16#A: CRC-16 (**SYS_BCC_CALCULATION_METHOD_CRC16**)

(2) Starting byte position for calculation (No. of bytes from **s2**)
0–F

(3) Starting byte position for storing results (No. of bytes from **d**)

(4) Conversion data
0: Binary data (CRC: 2 bytes/Not CRC: 1 byte)
1: ASCII data (2 bytes)

Note

If CRC-16 is specified as the calculation method, ASCII code cannot be specified for the conversion data.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if the number of specified bytes for the target data exceeds the limit of the specified data area.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if the number of specified bytes for the target data exceeds the limit of the specified data area.

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

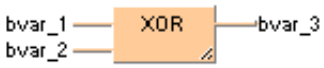
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | Start | BOOL | FALSE | |
| 1 | VAR | BCC_Calc_Method | INT | 2 | 0 = Addition |
| 2 | VAR | ASCII_String | STRING[32] | '%01#RCSX0000' | |
| 3 | VAR | BCC | WORD | 0 | result = 16#1D |

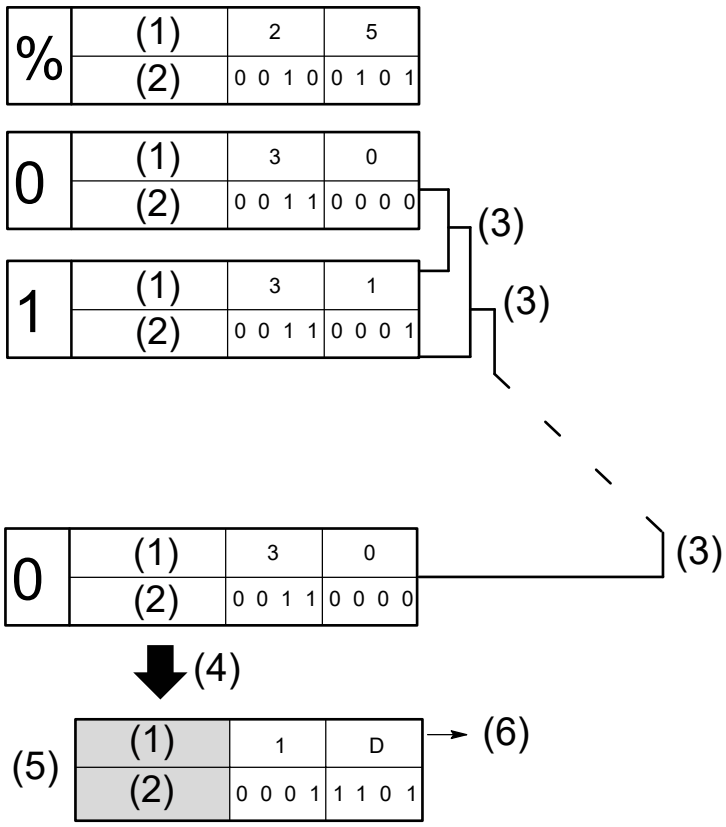
## LD body

A block check character calculation is performed on **ASCII_String** when **Start** turns to TRUE. The calculation method is exclusive OR. (Use this method when large amounts of data are transmitted).

How the BCC is calculated using the exclusive OR operation:

| bvar_1 | bvar_2 | bvar_3 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




(1) ASCII-HEX-code
(2) ASCII-BIN-code
(3) Exclusive ORing
(4) calculation
(5) Block Check Character (BCC)
(6) Calculation result (16#1D) is stored in **d**.

The ASCII BIN code bits of the first two characters are compared with each other to yield an 8-character exclusive OR operation result:

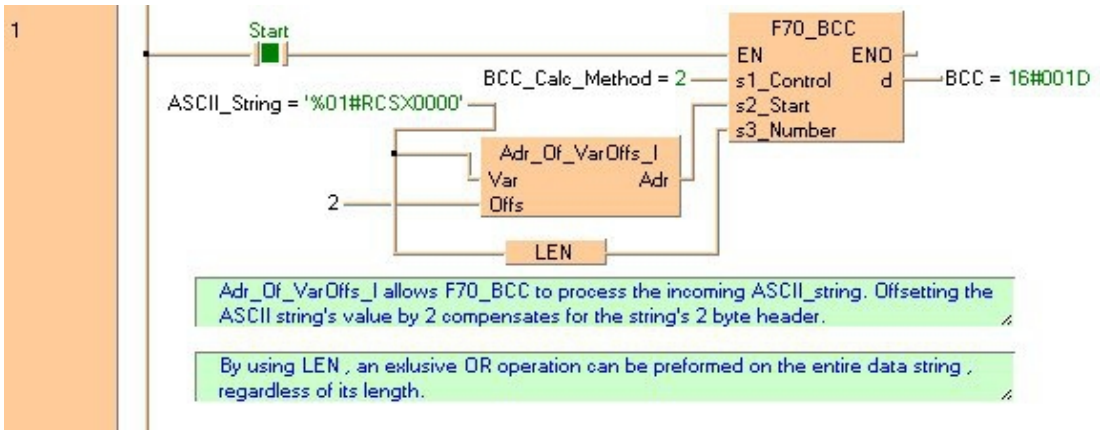
| Sign for comparison | ASCII BIN code |
|---|---|
| % | 00100101 |

| | |
|---|---|
| 0 | 00110000 |
| Exclusive OR result | 00010101 |

This result is then compared to the ASCII BIN code of the next character, i.e. "1".

| Sign for comparison | ASCII BIN code |
|---|---|
| Exclusive OR result | 00010101 |
| 1 | 00110001 |
| Next exclusive OR | 00100100 |

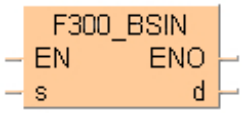
And so on until the final character is reached.



# F300_BSIN

BCD type sine operation

The function calculates the sine of **BCD** code angular data (input **s**) and stores the result (output **d**) as a **BCD** value in an array with three elements.




## Parameters

Input

**s** (WORD)

16-bit area where angle data is stored

Output

**d** ARRAY [0..2] of (WORD)

Result stored in 3 words

## Remarks

**BCD** values for input **s** lie in the area from 0° to 360° (16#0 to 16#360) in 1° steps. With this, output **d** can yield a result in the range of -1.0000 to 1.0000. The result is returned as follows:

ARRAY[0]

preceding sign (0 when input is +, 1 when input is -)

ARRAY[1]

whole number before the decimal point (0 or 1)

ARRAY[2]

numbers after the decimal point with 4 significant figures as a BCD value (16#0000 to 16#9999).

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if **s** is not a BCD value
- if **s** is not between 0° and 360°.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if **s** is not a BCD value
- if **s** is not between 0° and 360°.

**sys_bIsEqual** (turns to TRUE and remains TRUE)

if the result is 0.

**sys_bIsCarry** (turns to TRUE for one scan)

if the result is overflowed.

## Example

### POU header

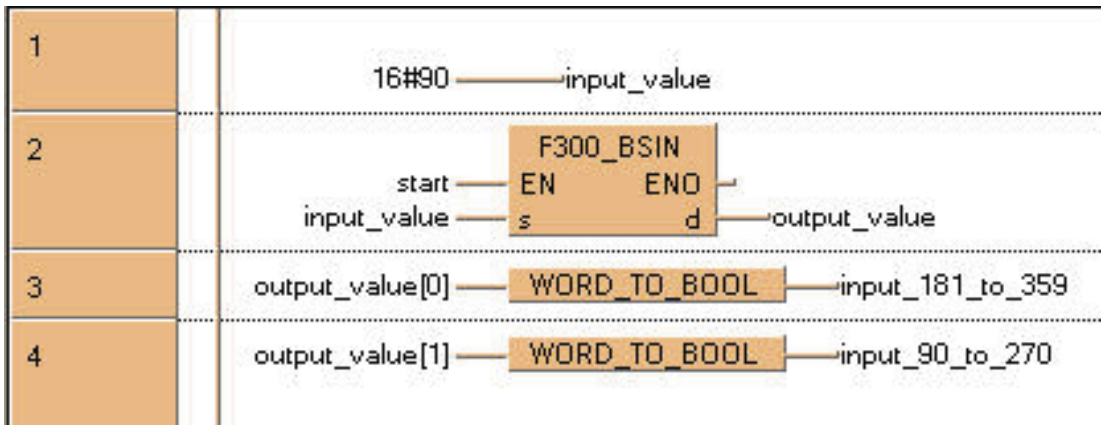
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | input_value | WORD | 0 | BCD value between |
| 2 | VAR | output_value | ARRAY [0..2] OF WORD | [3(0)] | number between -1.0000 and 1.0000 |
| 3 | VAR | input_181_to_359 | BOOL | FALSE | TRUE if input_value |
| 4 | VAR | input_90_or_270 | BOOL | FALSE | TRUE if input_value 90° or 270° |
| 5 | VAR | | | | |

In this example, the input variable **input_value** is declared. However, you can write a constant (e.g. 16#45 for 45°) directly at the input contact of the function.

### LD body

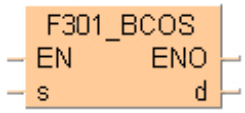
In the body, the value 90° is assigned to the variable **input_value**. When the variable **start** is set to TRUE, the function **F300_BSIN** is carried out. It stores the result in the variable **output_value**. If the **input_value** is between 181° and 359°, **output_value** has a minus sign. The function **WORD_TO_BOOL** sets the variable **input_181_to_359** to TRUE. With an **input_value** of 90° or 270°, the **output_value** is 1, which represents the value before the decimal point. If this is the case, then **WORD_TO_BOOL** sets the value of the variable **input_90_or_270** to TRUE.


1
16#90
input_value
2
F300_BSIN
start
EN
ENO
input_value
s
d
output_value
3
output_value[0]
WORD_TO_BOOL
input_181_to_359
4
output_value[1]
WORD_TO_BOOL
input_90_to_270

# F301_BCOS

BCD type cosine operation

The function calculates the cosine of **BCD** code angular data (input **s**) and stores the result (output **d**) as a **BCD** value in an array with three elements.




## Parameters

Input

**s** (WORD)

16-bit area where angle data is stored

Output

**d** ARRAY [0..2] of (WORD)

Result stored in 3 words

## Remarks

**BCD** values for input **s** lie in the area from 0° to 360° (16#0 to 16#360) in 1° steps. With this output **d** can yield a result in the range of -1.0000 to 1.0000. The result is returned as follows:

ARRAY[0]

preceding sign (0 when input is +, 1 when input is -)

ARRAY[1]

whole number before the decimal point (0 or 1)

ARRAY[2]

numbers after the decimal point with 4 significant figures as a BCD value (16#0000 to 16#9999).

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if **s** is not a BCD value
- if **s** is not between 0° and 360°.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if **s** is not a BCD value
- if **s** is not between 0° and 360°.

**sys_blsEqual** (turns to TRUE and remains TRUE)

if the result is 0.

**sys_blsCarry** (turns to TRUE for one scan)

if the result is overflowed.

## Example

### POU header

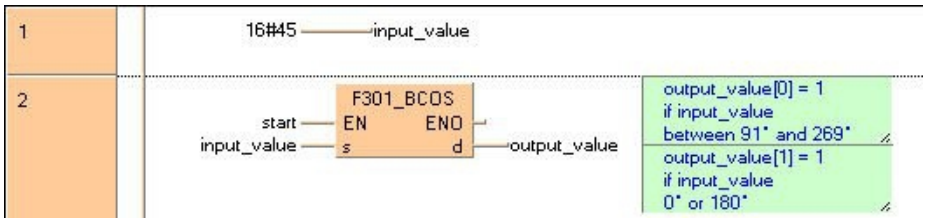
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | input_value | WORD | 0 | BCD value between |
| 2 | VAR | output_value | ARRAY [0..2] OF WORD | [3(0)] | number between -1.0000 and 1.0000<br>output_value [0] = +/- sign<br>output_value [1] = pre-decimal value<br>output_value [2] = post-decimal point values<br>result: here +0.7071 |
| 3 | VAR | | | | |

In this example, the input variable **input_value** is declared. However, you can write a constant (e.g. 16#45 for 45°) directly at the input contact of the function.

### LD body

In the body, the value 16#45° is assigned to the variable **input_value**. When the variable **start** is set to TRUE, the function is carried out. The result at output **d** is **output_value[0]** = 0, **output_value[1]** = 0, **output_value[2]** = 7071.



# F302_BTAN

BCD type tangent operation

The function calculates the tangent of **BCD** code angular data (input **s**) and stores the result (output **d**) as a **BCD** value in an array with three elements.


## Parameters

Input

**s** (WORD)

Area where angle data is stored

Output

**d** (ARRAY [0..2] of WORD)

Result stored in 3 words

## Remarks

**BCD** values for input **s** lie in the area from 0° to 360° (16#0 to 16#360) in 1° steps. With this output **d** yields a result in the range of -57.2900 to 57.2900. The result is returned as follows:

ARRAY[0]

preceding sign (0 when input is +, 1 when input is -)

ARRAY[1]

whole number before the decimal point (0 or 1)

ARRAY[2]

numbers after the decimal point with 4 significant figures as a BCD value (16#0000 to 16#9999).

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if **s** is not a BCD value
- if **s** is not between 0° and 360°.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if **s** is not a BCD value
- if **s** is not between 0° and 360°.

**sys_blsEqual** (turns to TRUE and remains TRUE)

if the result is 0.

**sys_blsCarry** (turns to TRUE for one scan)

if the result is overflowed.

## Example

### POU header

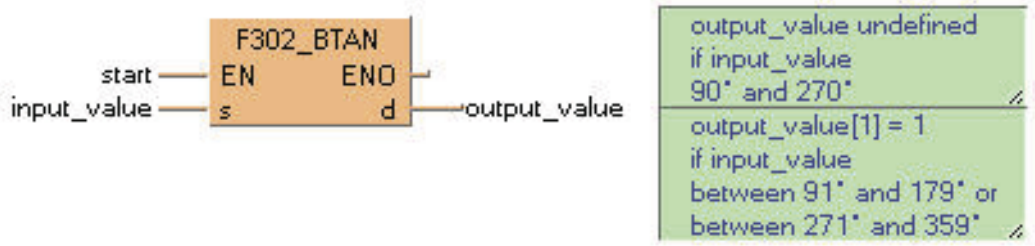
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | input_value | WORD | 16#89 | BCD value between number between -57.2900 and 57.2900 |
| 2 | VAR | output_value | ARRAY [0..2] OF WORD | [3(0)] | output_value[0] = +/- sign<br>output_value[1] = pre-decimal point values<br>output_value[2] = post-decimal point values<br>result: here +57.2899 |
| 3 | VAR | | | | |

In this example, the input variable **input_value** is declared. However, you can write a constant (e.g. 16#89 for 89°) directly at the input contact of the function.

### LD body

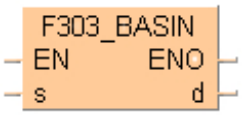
When the variable **start** is set to TRUE, the function is carried out. The **input_value** was initialized with the value 16#89 (89°) in the POU header. The result is written to the ARRAY **output_value**. Here in the first element of the ARRAY, the **output_value** = 16# (+ sign). In the second element, 16#57 represents the number before the decimal point, and 16#2899 comes after the decimal point in the third element.



# F303_BASIN

BCD type arcsine operation

The function calculates the arcsine of a **BCD** value that is entered at input **s** as an ARRAY with three elements. The result is returned as **BCD** angular data in the range of 0° to 360° (16#0 to 16#360) at output **d**.




## Parameters

Input

**s** ARRAY [0..2] of (WORD)

Area where angle data is stored

Output

**d** (WORD)

Result stored in 3 words

## Remarks

**BCD** values for input **s** lie in the area from -1.0000 to 1.0000. They are entered as follows:

ARRAY[0]

preceding sign (0 when input is +, 1 when input is -)

ARRAY[1]

whole number before the decimal point (0 or 1)

ARRAY[2]

numbers after the decimal point with 4 significant figures as a BCD value (16#0000 to 16#9999).

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if **s** is not a BCD value
- if **s** is not between -1.0000 and 1.0000

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if **s** is not a BCD value
- if **s** is not between -1.0000 and 1.0000

**sys_blsEqual** (turns to TRUE and remains TRUE)

if the result is 0.

**sys_blsCarry** (turns to TRUE for one scan)

if the result is overflowed.

## Example

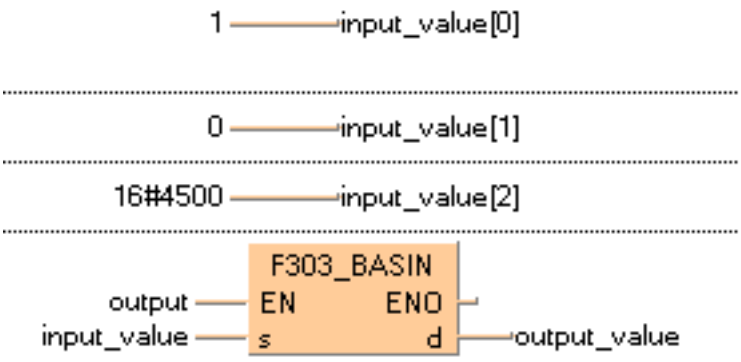
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | input_value | ARRAY [0..2] OF WORD | [3(0)] | number between -1.0000 and 1.0000 |
| 2 | VAR | output_value | WORD | 0 | BCD value between 16#0 and 16#360 (0° and 360°) result: here 16#333 |
| 3 | VAR | | | | |

### LD body

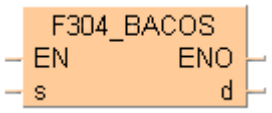
The first element of the ARRAY's **input_value** is given the value 1 (- sign). The second element has 0 as its whole number value, and in the third element 16#4500 is written as the value after the decimal point. When the variable **start** is set to TRUE, the function is carried out. The result for the **output_value** = 16#333 (333°).



# F304_BACOS

BCD type arccosine operation

The function calculates the arccosine of a **BCD** value that is entered at input **s** as an ARRAY with three elements. The result is returned as **BCD** angular data in the range of 0° to 360° (16#0 to 16#360) at output **d**.




## Parameters

Input

**s** ARRAY [0..2] of (WORD)

Area where angle data is stored in 3 words

Output

**d** (WORD)

Result

## Remarks

**BCD** values for input **s** lie in the area from -1.0000 to 1.0000. They are entered as follows:

ARRAY[0]

preceding sign (0 when input is +, 1 when input is -)

ARRAY[1]

whole number before the decimal point (0 or 1)

ARRAY[2]

numbers after the decimal point with 4 significant figures as a BCD value (16#0000 to 16#9999).

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if **s** is not a BCD value
- if **s** is not between -1.0000 and 1.0000

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if **s** is not a BCD value
- if **s** is not between -1.0000 and 1.0000

**sys_blsEqual** (turns to TRUE and remains TRUE)

if the result is 0.

**sys_blsCarry** (turns to TRUE for one scan)

if the result is overflowed.

## Example

### POU header

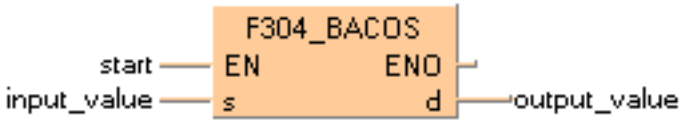
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | input_value | ARRAY [0..2] OF WORD | [2(0),16#8660] | number between -1.0000 and 1.0000 |
| 2 | VAR | output_value | WORD | 0 | BCD value between 16#0 and 16#360 (0° and 360°) result: here 16#30 |
| 3 | VAR | | | | |

### LD body

When the variable **start** is set to TRUE, the function is carried out.

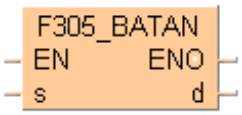
The **input_value** = 0 (+ sign) in the ARRAY's first element. 0 represents the whole in the second element, and the value after the decimal point is 8660. The function thus calculates the **output_value** = 16#30 (30°).



# F305_BATAN

BCD type arctangent operation

The function calculates the arctangent of a **BCD** value that is entered at input **s** as an ARRAY with three elements. The result is returned as **BCD** angular data in the range 0° to 90° (16#0 to 16#90) or 270° to 360° (16#270 to 16#360) at output **d**.




## Parameters

Input

**s** ARRAY [0..2] of (WORD)

Area where angle data is stored in 3 words

Output

**d** (WORD)

Result

## Remarks

**BCD** values for input **s** lie in the area from -9999.9999 to 9999.9999. They are entered as follows:

ARRAY[0]

preceding sign (0 when input is +, 1 when input is -)

ARRAY[1]

whole number before the decimal point (0 or 1)

ARRAY[2]

numbers after the decimal point with 4 significant figures as a BCD value (16#0000 to 16#9999).

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if **s** is not a BCD value

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if **s** is not a BCD value

**sys_bIsEqual** (turns to TRUE and remains TRUE)

if the result is 0.

**sys_bIsCarry** (turns to TRUE for one scan)

if the result is overflowed.

## Example

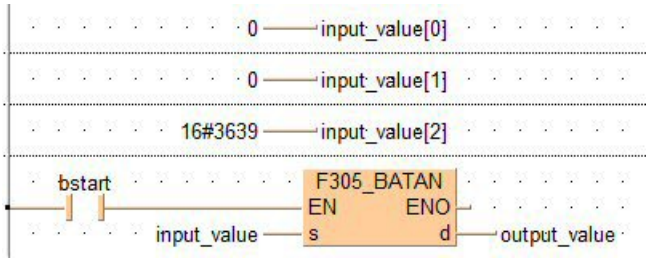
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | output | BOOL | FALSE | activates the function |
| 1 | VAR | input_value | ARRAY [0..2] OF WORD | [3(0)] | number between -9999.9999 and 9999.9999<br>input_value[0] = +/- sign<br>input_value[1] = pre-decimal point values<br>input_value[2] = post-decimal point values |
| 2 | VAR | output_value | WORD | 0 | |

### LD body

The first element of the ARRAY's **input_value** is given the value 0 (+ sign). 0 is the whole number value in the second element, and the third element has a value of 3639 after the decimal point. When the variable **start** is set to TRUE, the function is carried out. The result for the **output_value** = 16#20 (20°).



# 3 Bistable instructions

# R

Reset (a Boolean operand)

- Depending on the content of the accumulator, the operand defined in the operand field is reset or kept. TRUE in the accumulator resets the operand, i.e. the operand is set to 0 (FALSE).
- Valid operands for this operator must be of one of the following data types:BOOL
- Operator only available in IL programming language.

## Example

### POU header

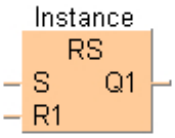
All input and output variables used for programming this function have been declared in the POU header.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | var_1 | INT | 0 | In/Output_1 |
| 1 | VAR | var_2 | INT | 0 | In/Output_2 |
| 2 | VAR | var_3 | INT | 0 | In/Output_3 |
| 3 | VAR | var_4 | INT | 0 | In/Output_4 |

# RS

Reset/set

The function block **RS** (reset/set) allows you to both reset and set an output.




## Parameters

Input

**S** (BOOL)

Set
The output **Q1** is set for each rising edge at **S** if RESET is not set.

**R1** (BOOL)

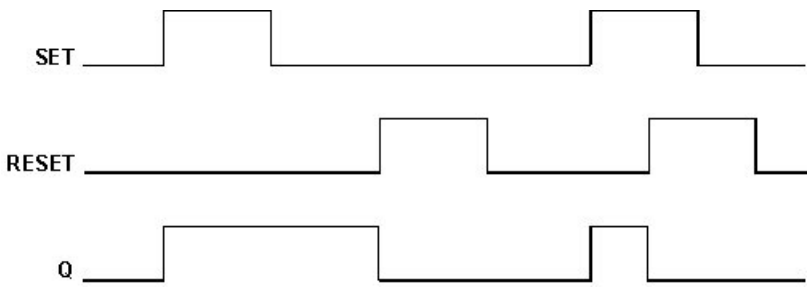
Reset
The output **Q1** is reset for each rising edge at **R1**.

Output

**Q1** (BOOL)

- set if a rising edge is detected at **S** and if **R1** is not set
- reset if a rising edge is detected at **R1**.
- reset if a rising edge is detected at both inputs.

## Time chart



## Example

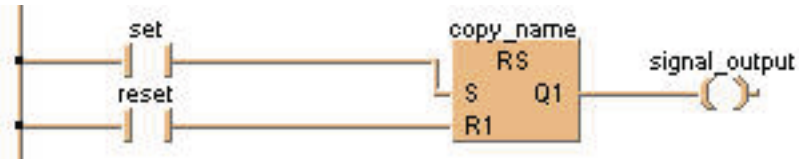
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | copy_name | RS | | under this identifier a copy of |
| 1 | VAR | set | BOOL | FALSE | set input |
| 2 | VAR | reset | BOOL | FALSE | reset input |
| 3 | VAR | signal_output | BOOL | FALSE | |

### LD body

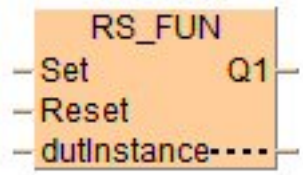
If **set** is set (status = TRUE) the **signal_output** will be set. If only **reset** is set, the **signal_output** will be **reset** (status = FALSE). If both **set** and **reset** are set, the **signal_output** will be **reset** to FALSE.



# RS_FUN

Reset/set

This is a user-defined function from a system function block. **RS_FUN** (reset/set) allows you to both reset and set an output.




## Parameters

Input

**Set** (BOOL)

Set
The output **Q1** is set for each rising edge at **S** if RESET is not set.

**Reset** (BOOL)

Reset
The output **Q1** is reset for each rising edge at **R1**.

Input/output

**dutInstance** (RS_FUN_INSTANCE_DUT)

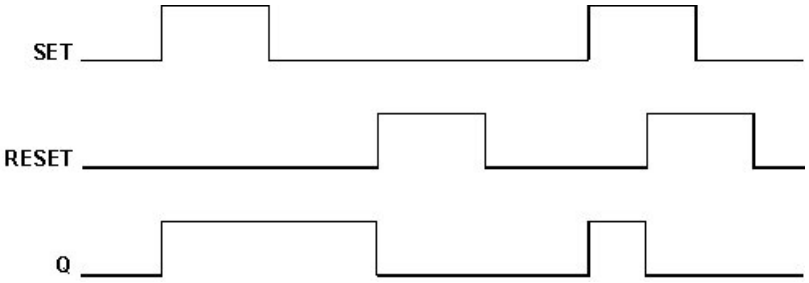
Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

Output

**Q1** (BOOL)

- set if a rising edge is detected at **Set** and if **Reset** is not set
- reset if a rising edge is detected at **Reset**.
- reset if a rising edge is detected at both inputs.

**Time chart**



# S

Set (a Boolean operand)

- Depending on the content of the accumulator, the operand defined in the operand field is set or kept. TRUE in the accumulator sets the operand, i.e. the operand is set to 1 (TRUE).
- Valid operands for this operator must be of one of the following data types:BOOL.
- Operator only available in IL programming language.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | var_1 | INT | 0 | In/Output_1 |
| 1 | VAR | var_2 | INT | 0 | In/Output_2 |
| 2 | VAR | var_3 | INT | 0 | In/Output_3 |
| 3 | VAR | var_4 | INT | 0 | In/Output_4 |

### POU body

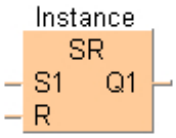
**var_1** and **var_2** can be of any valid data type, **var_3** and **var_4** have to be of type BOOL.

**var_3** is only reset (set to 0 or FALSE) if the accumulator is TRUE, i.e. if **var_1**=**var_2**

# SR

Set/reset

The function block **SR** (set/reset) allows you to both set and reset an output.




## Parameters

Input

**S1** (BOOL)

Set
The output **Q1** is set for each rising edge at **S1**

**R** (BOOL)

Reset
The output **Q1** is reset for each rising edge detected at **R**, except when **S1** is set (see time chart)

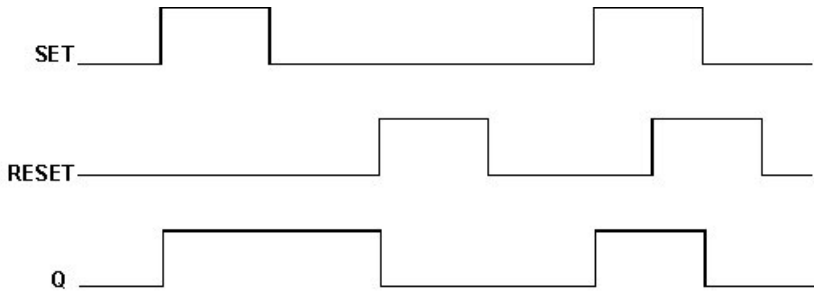
Output

**Q1** (BOOL)

Set or reset depending on inputs

- set if a rising edge is detected at **S1**
- reset if a rising edge is detected at **R** if **S1** is not set.
- set if a rising edge is detected at both inputs (**S1** and **R**).
- Upon initialising, **Q1** always has the status zero (reset).

## Time chart



## Example

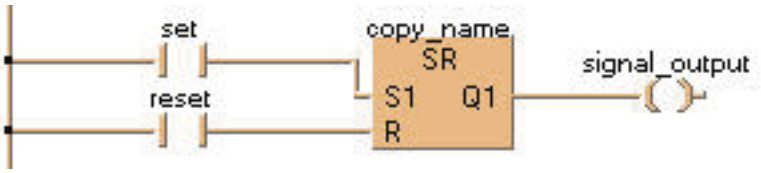
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | copy_name | SR | | under this identifier a copy of |
| 1 | VAR | set | BOOL | FALSE | set input |
| 2 | VAR | reset | BOOL | FALSE | reset input |
| 3 | VAR | signal_output | BOOL | FALSE | |

### LD body

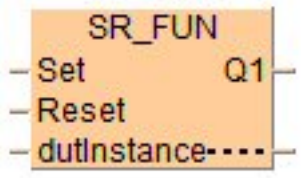
If **set** is set (status = TRUE), **signal_output** will be set. If only **reset** is set, the **signal_output** will be **reset** (status = FALSE). If both **set** and **reset** are set, **signal_output** will be set.



# SR_FUN

Set/reset

This is a user-defined function from a system function block. **SR_FUN** (set/reset) allows you to both set and reset an output.




## Parameters

Input

**Set** (BOOL)

Set

The output **Q1** is set for each rising edge at **Set**

**Reset** (BOOL)

Reset

The output **Q1** is reset for each rising edge detected at **Reset**, except when **S1** is set (see time chart)

Input/output

**dutInstance** (SR_FUN_INSTANCE_DUT)

Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

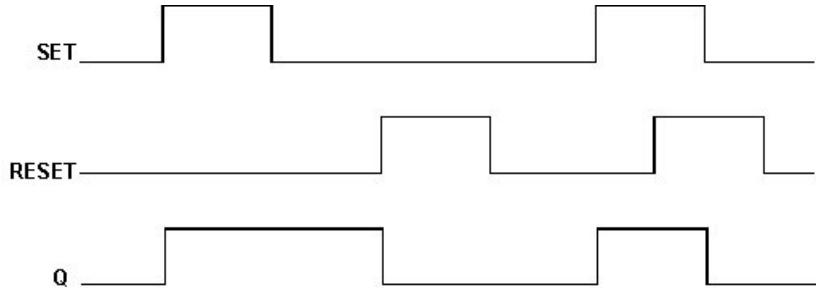
Output

**Q1** (BOOL)

Set or reset depending on inputs

- set if a rising edge is detected at **Set**
- reset if a rising edge is detected at **Reset** if **Set** is not set.
- set if a rising edge is detected at both inputs (**Set** and **Reset**).
- Upon initialising, **Q1** always has the status zero (reset).

**Time chart**



# KEEP

Serves as a relay with set and reset inputs

**KEEP** serves as a flip-flop with set and reset points.

When **SetTrigger** turns to TRUE, the specified flag turns to TRUE and maintains its condition. The flag turns to FALSE when **ResetTrigger** turns to TRUE. The flag's TRUE state is maintained until **ResetTrigger** turns to TRUE regardless of the TRUE or FALSE state of **SetTrigger**. If **SetTrigger** and **ResetTrigger** turn to TRUE simultaneously, priority is given to **ResetTrigger**.


## Parameters

Input

**Set Trigger** (BOOL)

Turns **Address** to TRUE

**Reset Trigger** (BOOL)

Turns **Address** to FALSE

Output

**Address** (BOOL)

The status of this flag (TRUE or FALSE) is kept.

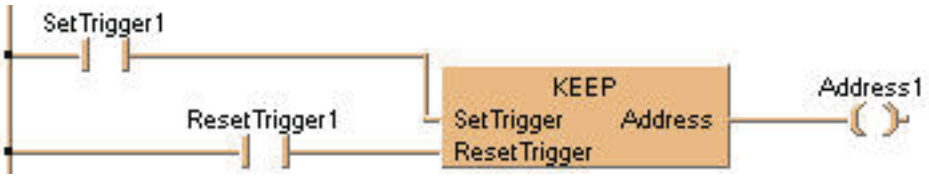
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | Set_trigger1 | BOOL | FALSE | if set_trigger is ON, the |
| 1 | VAR | Reset_trigger1 | BOOL | FALSE | if reset_trigger is ON, the |
| 2 | VAR | Address1 | BOOL | FALSE | output |

### LD body


SetTrigger1
ResetTrigger1
KEEP
SetTrigger
ResetTrigger
Address
Address1

# SET/

Set, reset output

- **SET**: When the execution conditions have been satisfied, the output is turned on, and the on status is retained.
- **RST**: When the execution conditions have been satisfied, the output is turned off, and the off status is retained.



## Remarks

- You can use flags with the same number as many times as you like with the **SET** and **RST** instructions. (Even if a total check is run, this is not handled as a syntax error.)
- When the **SET** and **RST** instructions are used, the output changes with each step during processing of the operation.
- To output a result while operation is still in progress, use a partial I/O update instruction (**F143_IORF**).
- The output destination of a **SET** instruction is held even during the operation of an **MC** instruction.
- The output destination of a **SET** instruction is reset when the mode is changed from RUN to PROG or when the power is turned off, except when a hold type internal flag is specified as the output destination.
- Placing a **DF** instruction (or specifying a rising edge in LD) before the **SET** and **RST** instructions ensures that the instruction is only executed at a rising edge.

## Flags

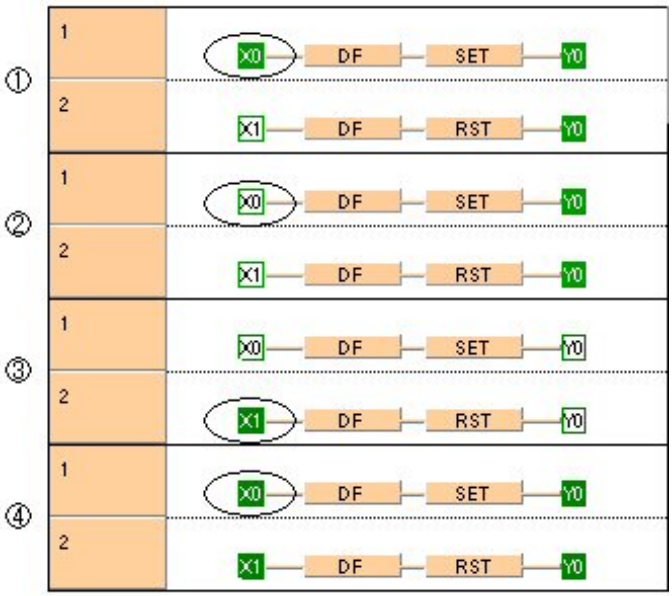
- Flags can be turned off using the **RST** instruction.
- Using the various flags with the **SET** and **RST** instructions does not result in double output.
- It is not possible to specify a pulse flag (P) as the output destination for a **SET** or **RST** instruction.

## Example

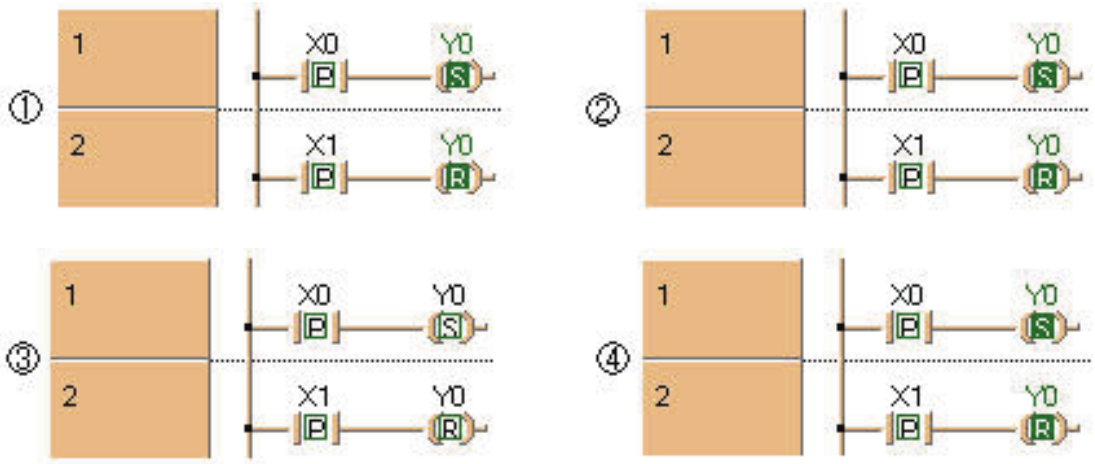
### POU header

Since addresses are assigned directly using FP addresses, no POU header is necessary.

### LD body




In ladder diagram, specify a rising edge in the contact and **SET** or **RESET** in the coil:

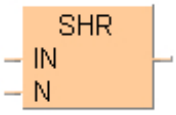


# 4 Bit-shift instructions

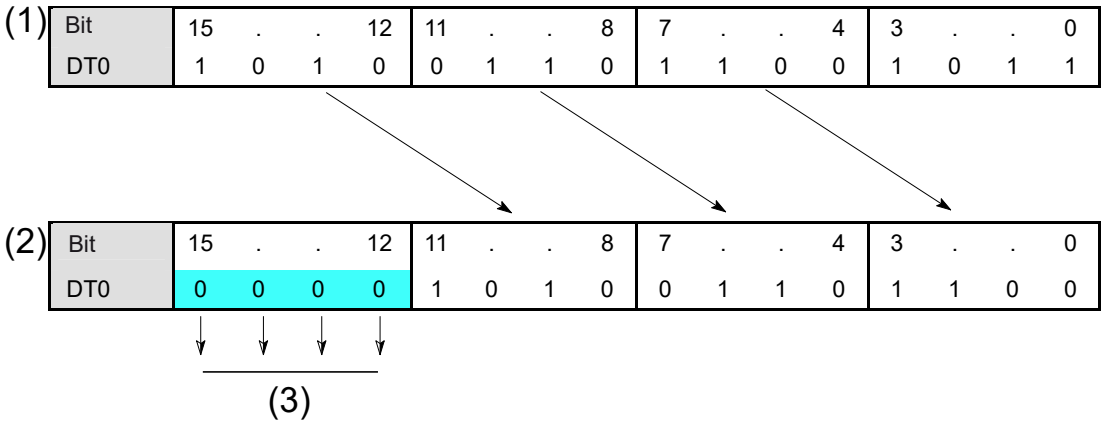
# SHR

Shift bits to the right

**SHR** shifts a bit value by a defined number of positions (N) to the right and fills the vacant positions with zeros.




### Bit shift to the right, zero-filled on left:




(1) source register
(2) target
(3) the 4 most significant bits are filled up with zeros

## Parameters

Input

**IN** (BOOL, WORD, DWORD)

1st input: input value

**N** (BOOL, WORD, DWORD)

2nd input: number of bits by which the input value is shifted to the right

Output

**Var_OUT** (BOOL, WORD, DWORD)

Output as input: result

Note

If the second input variable **N** (the number of bits to be shifted) is of the data type DWORD, then only the lower 16 bits are taken into account.

## Example

### POU header

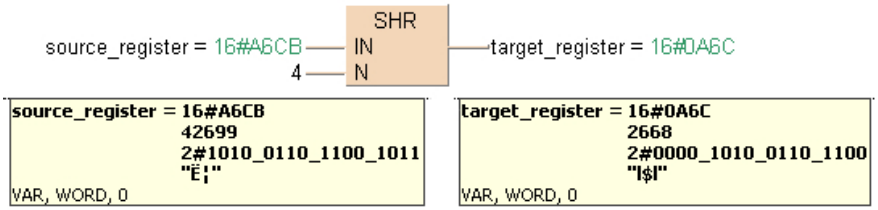
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | source_register | WORD | 0 |
| 1 | VAR | target_register | WORD | 0 |

This example uses variables. You can also use a constant for the input variable.

### LD body

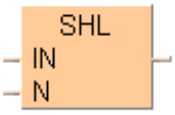
The last **N** bits (here 4) of **source_register** are right-shifted. The vacant positions on the left are filled with zeros. The result is written into **target_register**.



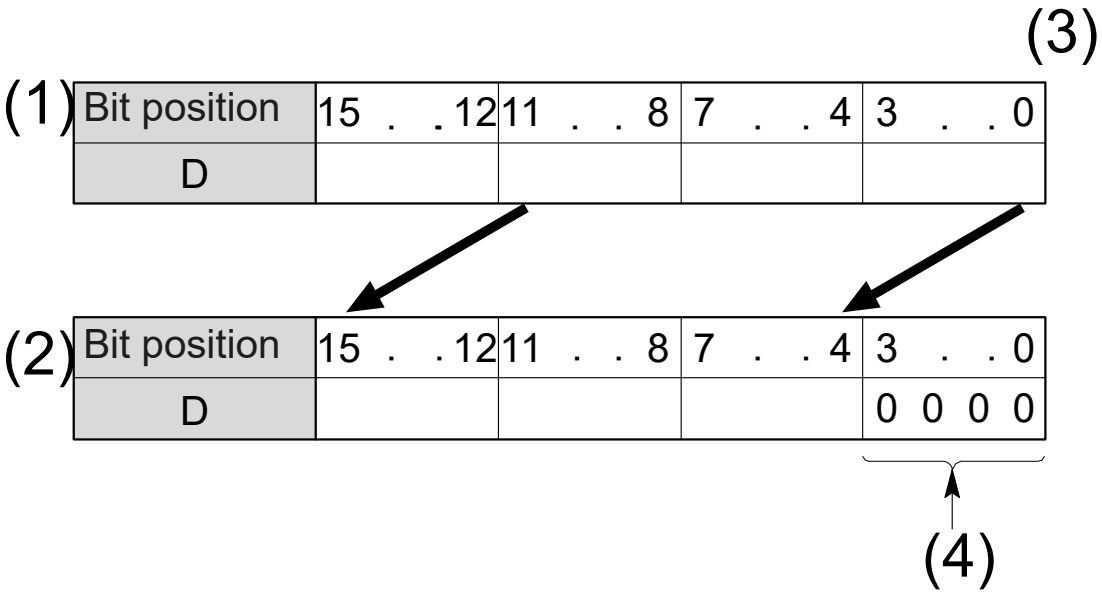
# SHL

Shift bits to the left

**SHL** shifts a bit value by a defined number of positions (N) to the left and fills the vacant positions with zeros.




**Bit shift to the left, zero-filled on right:**




(1) Bit position
(2) target register
(3) (n = 4 bit)
(4) n bits starting from bit position 0 are filled with 0s.

## Example

### POU header

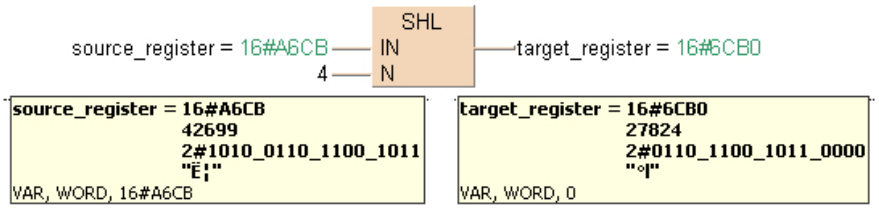
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | source_register | WORD | 0 |
| 1 | VAR | target_register | WORD | 0 |

This example uses variables. You can also use a constant for the input variable.

## LD body

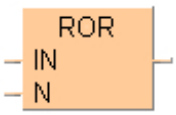
The first **N** bits (here 4) of **source_register** are left-shifted, the vacant positions on the right are filled with zeros. The result is written into **target_register**.



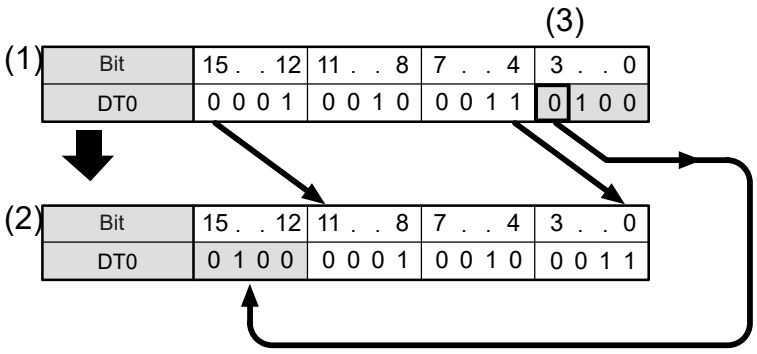
# ROR

Rotate N bits to the right

**ROR** rotates a defined number (N) of bits to the right.




### Example




(1) source register
(2) target register
(3) (n = 4 bit)

## Example

### POU header

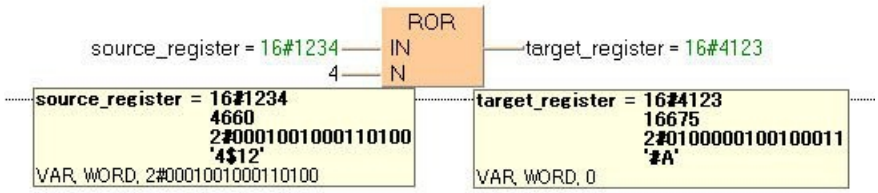
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | source_register | WORD | 0 |
| 1 | VAR | target_register | WORD | 0 |

This example uses variables. You can also use a constant for the input variable.

### LD body

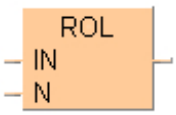
The first **N** bits (here N = 4) of **source_register** are right-rotated. The result will be written into **target_register**.


ROR
source_register = 16#1234
IN
target_register = 16#4123
4
N
source_register = 16#1234
4660
2#0001001000110100
'4$12'
VAR, WORD, 2#0001001000110100
target_register = 16#4123
16675
2#0100000100100011
'#A'
VAR, WORD, 0

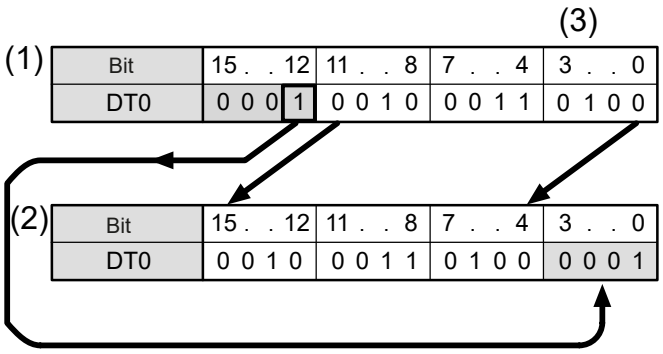
# ROL

Rotate N bits to the left

**ROL** rotates a defined number (N) of bits to the left.




### Example




(1) source register
(2) target register
(3) (n = 4 bit)

## Parameters

Input

**IN** (BOOL, WORD, DWORD)

Number of bits

**N** (BOOL, WORD, DWORD)

Number of bits

Output

**Var_OUT** (BOOL, WORD, DWORD)

Output as input: result

## Example

### POU header

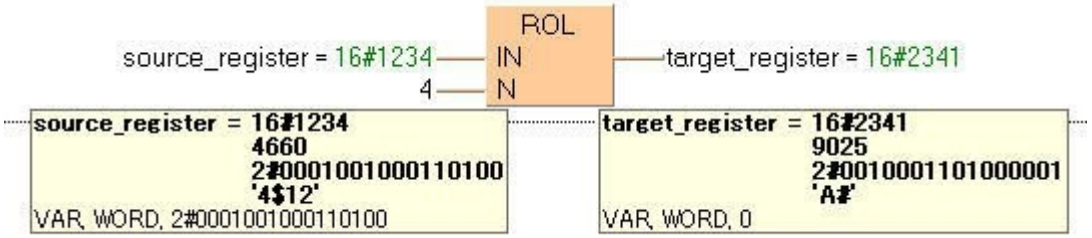
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | source_register | WORD | 0 |
| 1 | VAR | target_register | WORD | 0 |

This example uses variables. You can also use a constant for the input variable.

### LD body

The last **N** bits (here 4) of **source_register** are left-rotated. The result will be written in **target_register**.



## 4.5 FP instructions

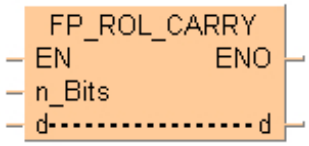
Tip

Advantages of FP instructions

# FP_ROL_CARRY

Rotate data to the left with carry flag

This FP instruction rotates the data specified by **d** to the left (to the higher bit position) by the number of bits specified by **n_Bits** and including the data of the carry flag if the trigger **EN** is TRUE.




## Parameters

Input

**n_Bits** (WORD, INT, UINT)

Number of bits
Values: 0–255

Input/output

**d** (WORD, DWORD)

Data to be rotated

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

if the area specified using the index modifier exceeds the limit.

**sys_blsCarry** (turns to TRUE for one scan)

- if **d**= 16-bit data and **n_Bits** is either 0 or a multiple of 17, no rotation occurs and the carry flag remains unchanged.
- if **d**= 32-bit data and **n_Bits** is either 0 or a multiple of 33, no rotation occurs and the carry flag remains unchanged.

## Example

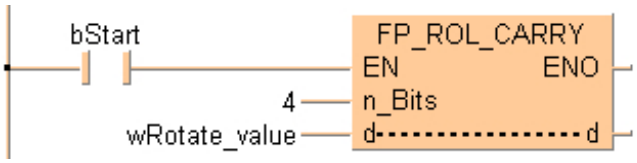
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | wRotate_value | WORD | 16#1234 | |

### LD body

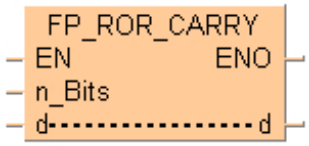
When the variable **bStart** is set to TRUE, the function is carried out.



# FP_ROR_CARRY

Rotate data to the right with carry flag

This FP instruction rotates the data specified by **d** to the right (to the lower bit position) by the number of bits specified by **n_Bits** and including the data of the carry flag if the trigger **EN** is TRUE.




## Parameters

Input

**n_Bits** (WORD, INT, UINT)

Number of bits
Values: 0–255

Input/output

**d** (WORD, DWORD)

Data to be rotated

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

if the area specified using the index modifier exceeds the limit.

**sys_blsCarry** (turns to TRUE for one scan)

- if **d**= 16-bit data and **n_Bits** is either 0 or a multiple of 17, no rotation occurs and the carry flag remains unchanged.
- if **d**= 32-bit data and **n_Bits** is either 0 or a multiple of 33, no rotation occurs and the carry flag remains unchanged.

## Example

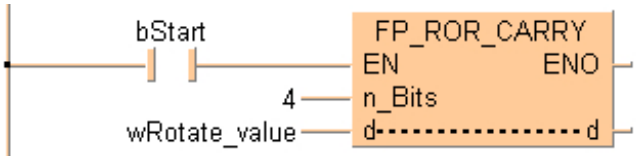
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | wRotate_value | WORD | 16#1234 | |

### LD body

When the variable **bStart** is set to TRUE, the function is carried out.



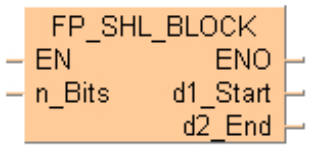
# FP_SHL_BLOCK

Shift data block to the left

This FP instruction shifts the data range specified by **d1_Start** and **d2_End** by the number of bits specified by **n_Bits** to the left (to the higher position) if the trigger **EN** is TRUE.

The vacant (empty) positions are filled with zeros.

- For **n_Bits** = 0, no shift takes place.
- For **n_Bits** = 16, a shift of one word as with **FP_WSHL_BLOCK** occurs.




## Parameters

Input

**n_Bits** (WORD, INT, UINT)

Number of bits (permissible range: 0–15)

Output

**d1_Start** (BOOL)

Starting address

**d2_End** (BOOL)

End address

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **d1_Start** > **d2_End**
- if **n_Bits** ≥ 16

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **d1_Start** > **d2_End**
- if **n_Bits** ≥ 16

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global variables

| | Class | Identifier | FP address | IEC address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | bStartAddress | R4 | %MX0.0.4 | BOOL | FALSE |
| 1 | VAR_GLOBAL | bEndAddress | R27 | %MX0.2.7 | BOOL | FALSE |

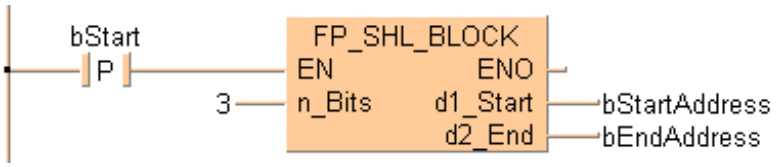
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR_EXTERNAL | bStartAddress | BOOL | FALSE | |
| 2 | VAR_EXTERNAL | bEndAddress | BOOL | FALSE | |

### LD body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out.



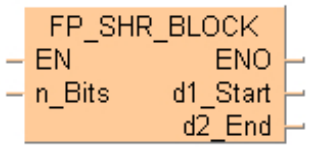
# FP_SHR_BLOCK

Shift data block to the right

This FP instruction shifts the data range specified by **d1_Start** and **d2_End** by the number of bits specified by **n_Bits** to the right (to the lower bit position) if the trigger **EN** is TRUE.

The vacant (empty) positions are filled with zeros.

- For **n_Bits** = 0, no shift takes place.
- For **n_Bits** = 16, a shift of one word as with **FP_WSHR_BLOCK** occurs.




## Parameters

Input

**n_Bits** (WORD, INT, UINT)

Number of bits (permissible range: 0–15)

Output

**d1_Start** (BOOL)

Starting address

**d2_End** (BOOL)

End address

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **d1_Start** > **d2_End**
- if **n_Bits** ≥ 16

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **d1_Start** > **d2_End**
- if **n_Bits** ≥ 16

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global variables

| | Class | Identifier | FP address | IEC address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | bStartAddress | R4 | %MX0.0.4 | BOOL | FALSE |
| 1 | VAR_GLOBAL | bEndAddress | R27 | %MX0.2.7 | BOOL | FALSE |

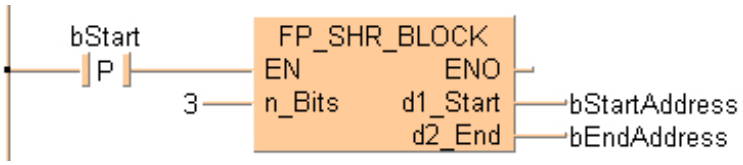
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR_EXTERNAL | bStartAddress | BOOL | FALSE | |
| 2 | VAR_EXTERNAL | bEndAddress | BOOL | FALSE | |

### LD body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out.

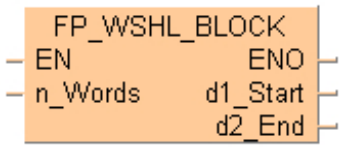


# FP_WSHL_BLOCK

Shift data block to the left

This FP instruction shifts the data range specified by **d1_Start** and **d2_End** by the number of words specified by **n_Words** to the left (to the higher position) if the trigger **EN** is TRUE.

The vacant (empty) positions are filled with zeros.




## Parameters

Input

**n_Words** (WORD, INT, UINT)

Number of words (permissible range: 0– 255)

Output

**d1_Start** (WORD)

Starting address

**d2_End** (WORD)

End address

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **d1_Start** > **d2_End**
- if **n_Words** is out of range

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **d1_Start** > **d2_End**
- if **n_Words** is out of range

## Example

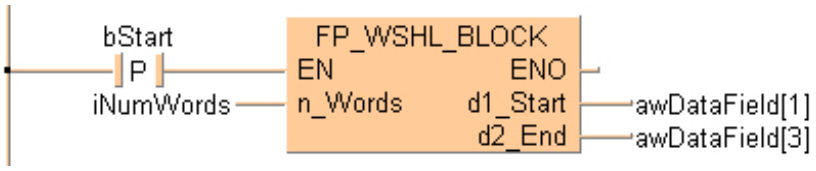
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | awDataField | ARRAY [0..3] OF WORD | [16#1001,16... | result after a 0->1 leading edge |
| 2 | VAR | iNumWords | INT | 0 | |

### LD body

When the variable **bStart** is set to TRUE, the function is carried out.

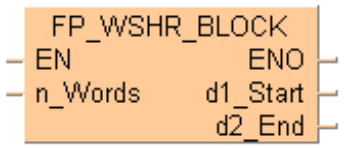


# FP_WSHR_BLOCK

Shift data block to the right

This FP instruction shifts the data range specified by **d1_Start** and **d2_End** by the number of words specified by **n_Words** to the right (to the lower position) if the trigger **EN** is TRUE.

The vacant (empty) positions are filled with zeros.




## Parameters

Input

**n_Words** (WORD, INT, UINT)

Number of words (permissible range: 0– 255)

Output

**d1_Start** (WORD)

Starting address

**d2_End** (WORD)

End address

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **d1_Start** >**d2_End**
- if **n_Words** is out of range

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **d1_Start** >**d2_End**
- if **n_Words** is out of range

## Example

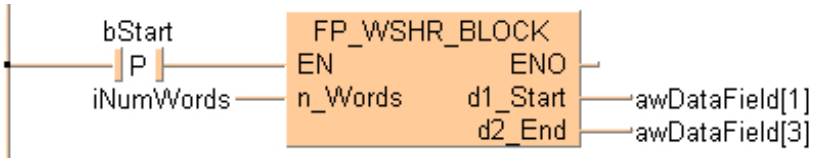
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | awDataField | ARRAY [0..3] OF WORD | [16#1001... | result after a 0->1 leading edge |
| 2 | VAR | iNumWords | INT | 0 | |

### LD body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out.



## 4.6 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

Advantages of IEC instructions

Advantages of FP instructions

# LSR

Left shift register

Shifts 1 bit of the specified data area **d_WR** to the left (to the higher bit position). When programming the **LSR** instruction, be sure to program the data input **DataInput**, shift **ShiftLeftTrigger** and reset triggers **Reset**.

Note

- The area available for this instruction is only the word internal relay (WR).
- Word internal relay (WR) number range, depends on the free area under “Extras” > “Options” > “Compile options” > “Address ranges”.


## Parameters

Input

**DataInput** (BOOL)

Specifies the state of new shift-in data:

- new shift-in data 1: when the input is ON
- new shift-in data 0: when the input is OFF

**ShiftLeftTrigger** (BOOL)

Shifts 1 bit to the left when the leading edge of the trigger is detected

**Reset** (BOOL)

Turns all the bits of the data area to 0 if the trigger is in the ON-state

Output

**d_WR** (WORD, INT, UINT)

Specified data area where data shift takes place

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

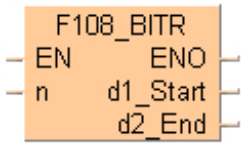
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Output | INT | 0 |
| 1 | VAR | DataInput | BOOL | FALSE |
| 2 | VAR | ShiftTrigger | BOOL | FALSE |
| 3 | VAR | ResetTrigger | BOOL | FALSE |

### LD body

# F108_BITR

Right shift of multiple bits of 16-bit data range

The function shifts the bits of a specified data range, whose beginning and end are specified by the outputs **d1** and **d2** to the right. The number of bits by which the data range is to be shifted to the right is specified by the value assigned at input **n**. The value may lie between 0 and 16. Bits cleared because of the shift become 0. When input **n** = 0, no shift takes place. When input **n** = 16, a shift of one WORD occurs, i.e. the same process takes place as with function F110_WHSL.




## Parameters

Input

**n** (INT)

Number of bits to be shifted

Output

**d1_Start** (WORD, INT, UINT)

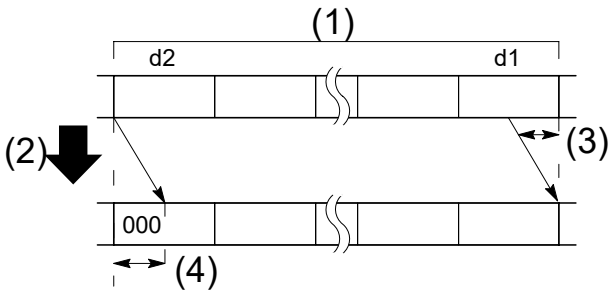
Starting 16-bit area

**d2_End** (WORD, INT, UINT)

Ending 16-bit area

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction:
- The addresses of the variables at inputs **d1_Start** and **d2_End** have to have the same address type.




(1) Specified data range
(2) start: ON
(3) n bits are shifted out
(4) n bits

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if the address of the variables at the outputs **d1_Start** > **d2_End** or the value at input is **n**≥ 16.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if the address of the variables at the outputs **d1_Start** > **d2_End** or the value at input is **n**≥ 16.

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | data_field | ARRAY [0..2] OF WORD | [16#1234,16#ABCD,16#5678] | Arbitrarily large data field, result: after a<br>0->1 leading edge of start<br>data_field[0] = 16#D123<br>data_field[1] = 16#8ABC<br>data_field[2] = 16#0567 |
| 2 | VAR | number_bits | INT | 4 | |
| 3 | VAR | | | | |

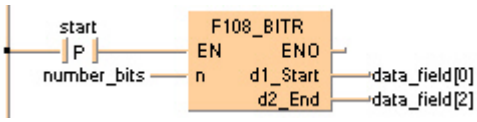
In this example, the input variable **number_bits** is declared. However, you can write a constant directly at the input contact of the function instead.

## POU body

When the variable **start** changes from FALSE to TRUE, the function is carried out.

It shifts out 4 bits (corresponds to one position in a hexadecimal representation) to the right. The 4 bits in **data_field[2]** resulting from the shift are filled with zeros.

## LD body

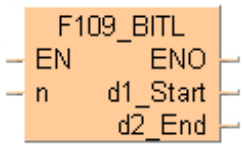

start
P
number_bits
F108_BITR
EN
ENO
n
d1_Start
d2_End
data_field[0]
data_field[2]

# F109_BITL

Left shift of multiple bits of 16-bit data range

The function shifts the bits of a specified data range, whose beginning and end are specified by the outputs **d1** and **d2** to the left. The number of bits by which the data range is to be shifted to the left is specified by the value assigned at input **n**. The value may lie between 0 and 16. Bits cleared because of the shift become 0.

- When input **n** = 0, no shift takes place.
- When input **n**= 16, a shift of one word occurs, i.e. the same process takes place as with function **F111_WSHL**.




## Parameters

Input

**n** (INT)

Number of bits to be shifted

Output

**d1_Start** (WORD, INT, UINT)

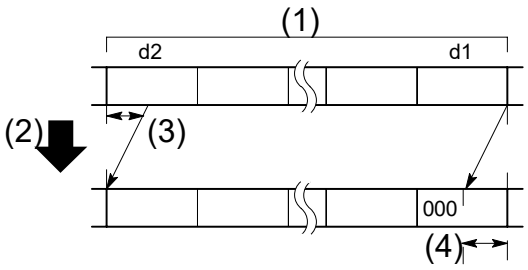
Starting 16-bit area

**d2_End** (WORD, INT, UINT)

Ending 16-bit area

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction:
- The addresses of the variables at inputs **d1** and **d2** have to have the same address type.




(1) Specified data range
(2) start: ON
(3) ending n bits are shifted out
(4) n bits

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if the address of the variables at the outputs **d1_Start** > **d2_End** or the value at input is **n**≥ 16.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if the address of the variables at the outputs **d1_Start** > **d2_End** or the value at input is **n**≥ 16.

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

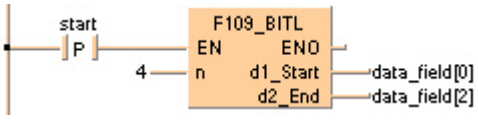
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | data_field | ARRAY [0..2] OF WORD | [16#1234,16#ABCD,16#5678] | Arbitrarily large data field, result: after a 0->1 leading edge of start: data_field[0] = 16#2340 data_field[1] = 16#BCD1 data_field[2] = 16#678A |
| 2 | VAR | | | | |

## POU body

When the variable **start** changes from FALSE to TRUE, the function is carried out.

It shifts out 4 bits (corresponds to one position in a hexadecimal representation) to the left. The 4 bits in **data_field[0]** resulting from the shift are filled with zeros. At input **n** the constant 4 is assigned directly to the function. You may, however, declare an input variable in the POU header instead.

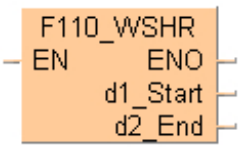
### LD body


start
P
F109_BITL
EN
ENO
4
n
d1_Start
data_field[0]
d2_End
data_field[2]

# F110_WSHR

Right shift of one word (16 bits) of 16-bit data range

Shifts one word (16 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the right (to the lower word address) if the trigger **EN** is in the ON-state.




## Parameters

Output

**d1_Start** (WORD, INT, UINT)

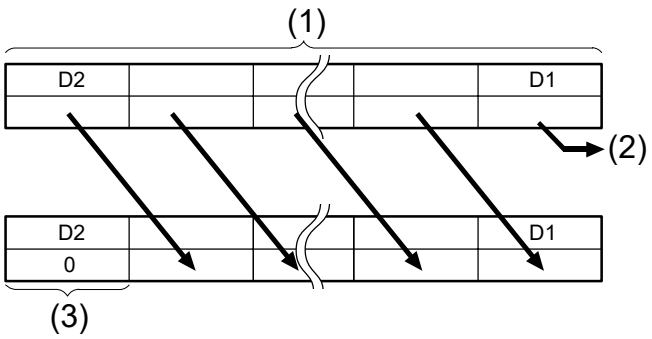
Starting 16-bit area

**d2_End** (WORD, INT, UINT)

Ending 16-bit area

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_WSHR_BLOCK**
- When one word (16 bits) is shifted to the right, the starting word is shifted out and the data in the ending word becomes 0.
- **d1_Start** and **d2_End** should be:
  - the same type of operand
  - **d1_Start≤d2_End**
- The variables **d1_Start** and **d2_End** have to be of the same data type.




(1) Specified data range
(2) The starting word is shifted out
(3) The data in the ending word becomes 0

## Example

### POU header

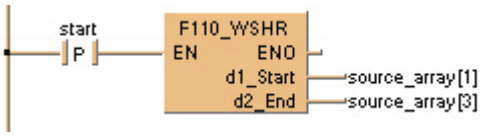
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | source_array | ARRAY [0..3] OF INT | [2,3,4,5] | result after a 0->1 leading edge from start: [2,4,5,0] |
| 2 | VAR | | | | |

### POU body

When the variable **start** changes from FALSE to TRUE, the function is carried out.

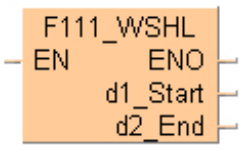
### LD body



# F111_WSHL

Left shift of one word (16 bits) of 16-bit data range

Shifts one word (16 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the left (to the higher word address) if the trigger **EN** is in the ON-state.




## Parameters

Output

**d1_Start** (WORD, INT, UINT)

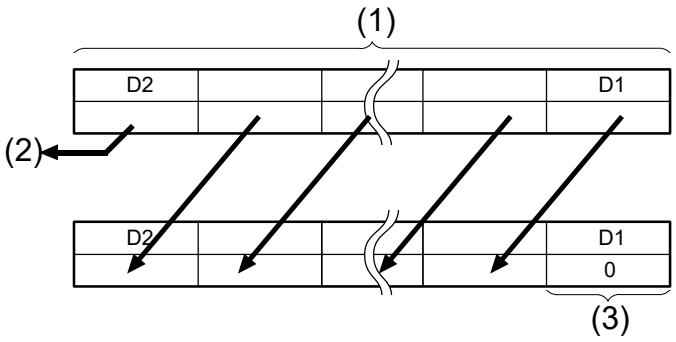
Starting 16-bit area

**d2_End** (WORD, INT, UINT)

Ending 16-bit area

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction:
- When one word (16 bits) is shifted to the left, the ending word is shifted out and the data in the starting word becomes 0.
- The variables **d1_Start** and **d2_End** have to be of the same data type.
- **d1_Start** and **d2_End** should be:
  - the same type of operand
  - **d1_Start** ≤ **d2_End**




(1) Specified data range
(2) The ending word is shifted out
(3) The data in the starting word becomes 0

## Example

### POU header

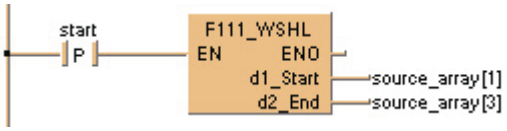
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | source_array | ARRAY [0..3] OF INT | [2,3,4,5] | result after a 0->1 leading edge from start: [2,0,3,4] |
| 2 | VAR | | | | |

### POU body

When the variable **start** changes from FALSE to TRUE, the function is carried out.

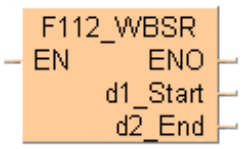
### LD body



# F112_WBSR

Right shift of one hex. digit (4 bits) of 16-bit data range

Shifts one hexadecimal digit (4 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the right (to the lower digit position) if the trigger **EN** is in the ON-state.




## Parameters

Output

**d1_Start** (WORD, INT, UINT)

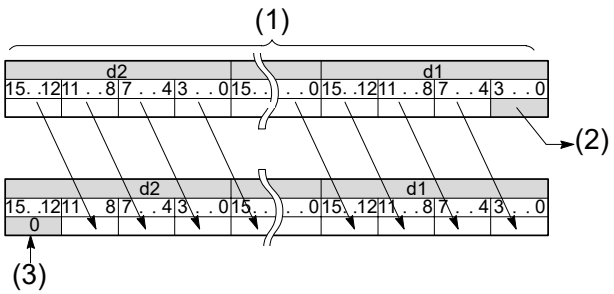
Starting 16-bit area

**d2_End** (WORD, INT, UINT)

Ending 16-bit area

## Remarks

- When one hexadecimal digit (4 bits) is shifted to the right:
  - the data in the lower hexadecimal digit (bit position 0 to 3) of the 16-bit data specified by **d1_Start** is shifted out.
  - the data in the higher hexadecimal digit (bit position 12 to 15) of the 16-bit data specified by **d2_End** becomes 0.
- **d1_Start** and **d2_End** should be:
  - the same type of operand
  - **d1_Start** ≤ **d2_End**
- The variables **d1_Start** and **d2_End** have to be of the same data type.




(1) Specified data range
(2) The data in the lower hexadecimal digit (bit positions 0 to 3) is shifted out
(3) The higher hexadecimal digit (bit positions 12 to 15) becomes 0

## Example

### POU header

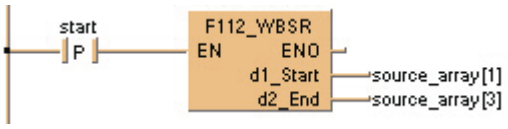
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | source_array | ARRAY [0..3] OF WORD | [16#3456,16#9012,16#5678,16#1234] | result after a 0->1 leading edge from start: [16#3456,16#8901, 16#4567,16#0123] |
| 2 | VAR | | | | |

### POU body

When the variable **start** changes from FALSE to TRUE, the function is carried out.

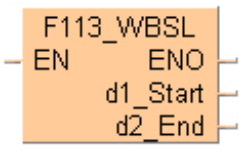
### LD body



# F113_WBSL

Left shift of one hex. digit (4 bits) of 16-bit data range

Shifts one hexadecimal digit (4 bits) of the data range specified by **d1** (starting) and **d2** (ending) to the left (to the higher digit position) if the trigger **EN** is in the ON-state.




## Parameters

Output

**d1_Start** (WORD, INT, UINT)

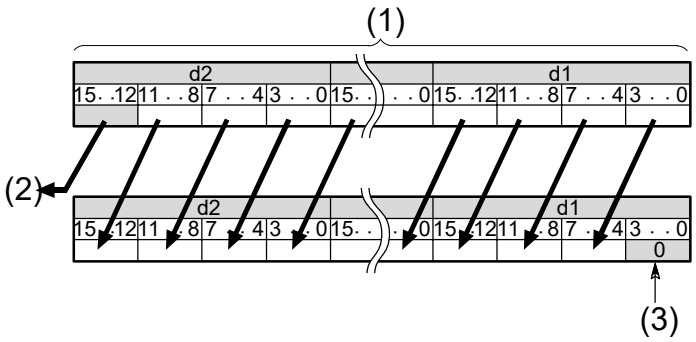
Starting 16-bit area

**d2_End** (WORD, INT, UINT)

Ending 16-bit area

## Remarks

- When one hexadecimal digit (4 bits) is shifted to the left,
  - the data in the higher hexadecimal digit (bit position 12 to 15) of the 16-bit data specified by **d2** is shifted out.
  - the data in the lower hexadecimal digit (bit position 0 to 3) of the 16-bit data specified by **d1** becomes 0.
- **d1** and **d2** should be:
  - the same type of operand
  - **d1** ≤ **d2**
- The variables **d1** and **d2** have to be of the same data type.




(1) Specified data range
(2) The data in the higher hexadecimal digit (bit positions 12 to 15) is shifted out
(3) The lower hexadecimal digit (bit positions 0 to 3) becomes 0

## Example

### POU header

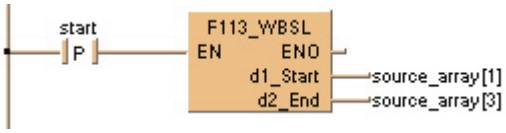
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | source_array | ARRAY [0..3] OF WORD | [16#3456,16#9012,16#5678,16#1234] | result after a 0->1 leading edge from start: [16#3456,16#0120, 16#6789,16#2345] |
| 2 | VAR | | | | |

### POU body

When the variable **start** changes from FALSE to TRUE, the function is carried out.

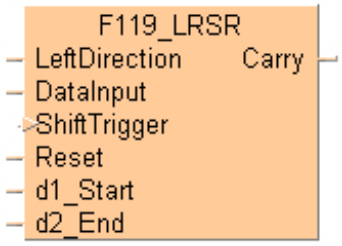
### LD body



# F119_LRSR

LEFT/RIGHT shift register

Shifts 1 bit of the 16-bit data range to the left or to the right.




## Parameters

Input

**LeftDirection** (BOOL)

Left/right trigger; specifies the direction of the shift-out:

- TRUE: shifting out to the left
- FALSE: shifting out to the right

**DataInput** (BOOL)

Specifies the new shift-in data.

- new shift-in data = TRUE = 1: when the data input is in the TRUE-state.
- new shift-in data = FALSE = 0: when the data input is in the FALSE-state.

**ShiftTrigger** (BOOL)

Activates shift
Shifts 1 bit to the left or right when the rising edge of the trigger is detected (FALSE→TRUE).

**Reset** (BOOL)

Turns all the bits of the data range specified by **d1_Start** and **d2_End** to 0 if this trigger is in the TRUE-state.
Resets data in area specified by **d1_Start** and **d2_End** to 0

**d2_End** (WORD, INT, UINT)

Ending 16-bit area

**d1_Start** (WORD, INT, UINT)

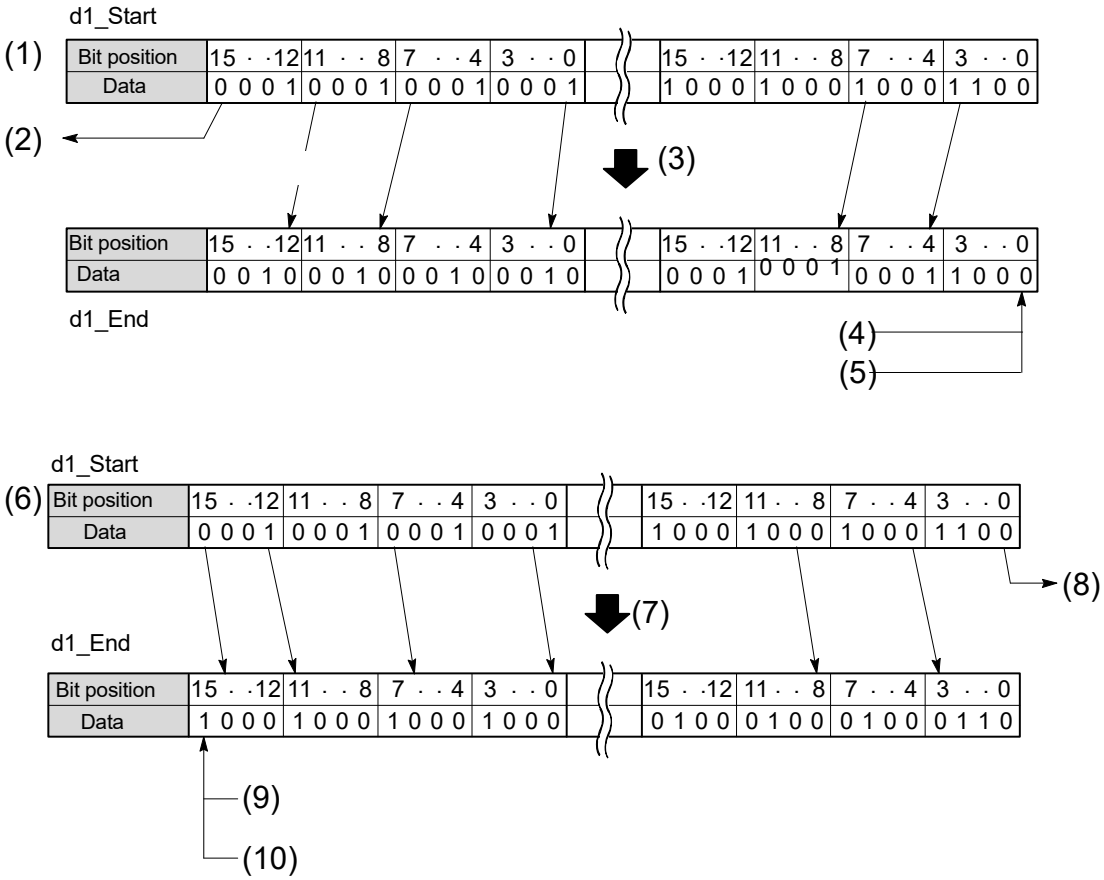
Starting 16-bit area

Output

**Carry** (BOOL)

Bit shifted out

## Example




(1) Left shift operation
(2) Shifted-out bit is transferred to R9009 (carry flag)
(3) LeftDirection: ON; Shift Trigger: OFF, ON
(4) When DataInput turns on, “1” is shifted into bit position 0.
(5) When DataInput turns off, “0” is shifted into bit position 0.
(6) Right shift operation
(7) LeftDirection:OFF; ShiftTrigger: OFF, ON
(8) Shifted-out bit is transferred to R9009 (carry flag).
(9) When DataInput turns on, “1” is shifted´ into bit position 15.
(10) When DataInput turns off, “0” is shifted´ into bit position 15.

## Remarks

- The variables **d1_Start** and **d2_End** have to be of the same data type.
- This function does not require a variable at the output **Carry**.
- Left/right shift is a shift register which shifts 1 bit of the specified data area to the left (to the higher bit position) or to the right (to the lower bit position).

## Example

### POU header

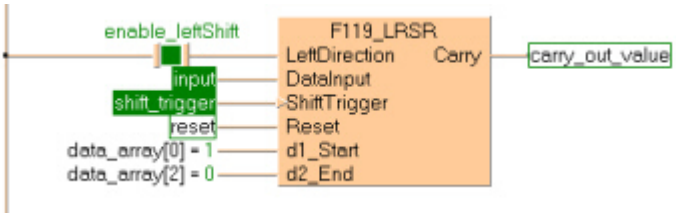
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | data_array | ARRAY [0..2] OF INT | [2#00000... | |
| 1 | VAR | enable_leftShift | BOOL | FALSE | function shifts left if TRUE, |
| 2 | VAR | reset | BOOL | FALSE | if TRUE, the whole array |
| 3 | VAR | input | BOOL | TRUE | specifies the new shift-in data |
| 4 | VAR | shift_trigger | BOOL | FALSE | activates the function at a 0->1 |
| 5 | VAR | carry_out_value | BOOL | FALSE | result after a 0->1 leading edge |

### POU body

When the variable **enable_leftShift** is set to TRUE, the function shifts left, else it shifts right.

### LD body

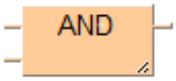


# 5 Bitwise Boolean instructions

# AND

Logical AND operation

The content of the accumulator is connected with the operand defined in the operand field by a logical **AND** operation. The result is transferred to the accumulator.




## Parameters

Input

**Unnamed input** (BOOL, WORD, DWORD)

1st input: element 1 of logical **AND** operation

**Unnamed input** (BOOL, WORD, DWORD)

2nd input: element compared to input 1

Output

**Unnamed output** (BOOL, WORD, DWORD)

Output as input: result

## Remarks

- All operands must be of the same data type.
- This function can be expanded to a maximum of 28 input contacts, see also modifying elements.

## Truth table:

| | Input 1 | Input 2 | Output |
|---|---|---|---|
| Signal | 0 | 0 | 0 |
| | 0 | 1 | 0 |
| | 1 | 0 | 0 |
| | 1 | 1 | 1 |

## Example

### POU header

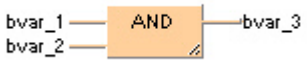
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | var_1 | BOOL | FALSE | Input_1 |
| 1 | VAR | var_2 | BOOL | FALSE | Input_2 |
| 2 | VAR | var_3 | BOOL | FALSE | Input_3 |
| 3 | VAR | var_4 | BOOL | FALSE | Output |

### POU body

**bvar_1** will be logically AND-linked with **bvar_2**. The result will be written into the output variable **bvar_3**.

### LD body



# ANDN

AND NOT Connection

The content of the accumulator is connected with the inverted operand defined in the operand field by a logical AND operation. The result is transferred to the accumulator.

## Remarks

- Operator only available in IL programming language.
- All operands must be of the same data type.
- Valid operands for this operator must be of one of the following data types: (BOOL, WORD, DWORD)

## Truth table:

| | Input 1 | Input 2 | Output |
|---|---|---|---|
| Signal | 0 | 0 | 1 |
| | 0 | 1 | 1 |
| | 1 | 0 | 1 |
| | 1 | 1 | 0 |

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | var_1 | BOOL | FALSE | Input_1 |
| 1 | VAR | var_2 | BOOL | FALSE | Input_2 |
| 2 | VAR | var_3 | BOOL | FALSE | Output |

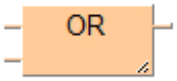
### IL body

| | | |
|---|---|---|
| LD | **var_1** | (* Load **var_1** in accu *) |
| ANDN | **var_2** | (* Perform an AND of accu with inverted **var_2**; store result in accu *) |
| ST | **var_3** | (* Store accu in **var_3** *) |

# OR

OR Connection

The content of the accumulator is connected with the operand defined in the operand field by a logical **OR** operation. The result is transferred to the accumulator.




## Parameters

Input

**Unnamed input** (BOOL, WORD, DWORD)

1st input: element 1 of logical OR operation

**Unnamed input** (BOOL, WORD, DWORD)

2nd input: element compared to input 1

Output

**Unnamed output** (BOOL, WORD, DWORD)

Output as input: result

## Remarks

- All operands must be of the same data type.
- This function can be expanded to a maximum of 28 input contacts, see also modifying elements.

## Truth table:

| | Input 1 | Input 2 | Output |
|---|---|---|---|
| Signal | 0 | 0 | 0 |
| | 1 | 0 | 1 |
| | 0 | 1 | 1 |
| | 1 | 1 | 1 |

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | var_1 | BOOL | FALSE | Input_1 |
| 1 | VAR | var_2 | BOOL | FALSE | Input_2 |
| 2 | VAR | var_3 | BOOL | FALSE | Input_3 |
| 3 | VAR | var_4 | BOOL | FALSE | Output |

# ORN

OR NOT Connection

The content of the accumulator is connected with the inverted operand defined in the operand field by a logical OR operation. The result is transferred to the accumulator.

## Remarks

- Operator only available in IL programming language.
- All operands must be of the same data type.
- Valid operands for this operator must be of one of the following data types: (BOOL, WORD, DWORD)

## Truth table:

| | Input 1 | Input 2 | Output |
|---|---|---|---|
| Signal | 0 | 0 | 1 |
| | 1 | 0 | 0 |
| | 0 | 1 | 0 |
| | 1 | 1 | 0 |

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | var_1 | BOOL | FALSE | Input_1 |
| 1 | VAR | var_2 | BOOL | FALSE | Input_2 |
| 2 | VAR | var_3 | BOOL | FALSE | Output |

### IL body

| | | |
|---|---|---|
| LD | **var_1** | (* Load **var_1** in accu *) |
| ORN | **var_2** | (* Perform an OR of accu with inverted var_2; store result in accu *) |
| ST | **var_3** | (* Store accu in **var_3** *) |

# XOR

Exclusive OR operation

The content of the accumulator is connected with the operand defined in the operand field by a logical **XOR** operation. The result is transferred to the accumulator.


## Parameters

Input

**Unnamed input** (BOOL, WORD, DWORD)

1st input: element 1 of logical **XOR** operation

**Unnamed input** (BOOL, WORD, DWORD)

2nd input: element compared to input 1

Output

**Unnamed output** (BOOL, WORD, DWORD)

Output as input: result

## Remarks

- All operands must be of the same data type.
- This function can be expanded to a maximum of 28 input contacts, see also modifying elements.

## Truth table:

| | Input 1 | Input 2 | Output |
|---|---|---|---|
| Signal | 0 | 0 | 0 |
| | 1 | 0 | 1 |
| | 0 | 1 | 1 |
| | 1 | 1 | 0 |

## Example

### POU header

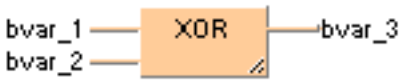
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | var_1 | BOOL | FALSE | Input_1 |
| 1 | VAR | var_2 | BOOL | FALSE | Input_2 |
| 2 | VAR | var_3 | BOOL | FALSE | Input_3 |
| 3 | VAR | var_4 | BOOL | FALSE | Output |

### POU body

The Boolean variables **bvar_1** and **bvar_2** are logically EXCLUSIVE-OR linked and the result is written in **bvar_3**.

### LD body



# XORN

Exclusive OR NOT Connection

The content of the accumulator is connected with the inverted operand defined in the operand field by a logical XOR operation. The result is transferred to the accumulator.

## Remarks

- Operator only available in IL programming language.
- All operands must be of the same data type.
- Valid operands for this operator must be of one of the following data types: (BOOL, WORD, DWORD)

## Truth table:

| | Input 1 | Input 2 | Output |
|---|---|---|---|
| Signal | 0 | 0 | 1 |
| | 1 | 0 | 0 |
| | 0 | 1 | 0 |
| | 1 | 1 | 1 |

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | var_1 | BOOL | FALSE | Input_1 |
| 1 | VAR | var_2 | BOOL | FALSE | Input_2 |
| 2 | VAR | var_3 | BOOL | FALSE | Output |

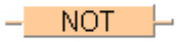
### IL body

| | | |
|---|---|---|
| LD | **var_1** | (* Load **var_1** in accu *) |
| XORN | **var_2** | (* Perform an **XOR** of accu with inverted **var_2**; store result in accu *) |
| ST | **var_3** | (* Store accu in **var_3** *) |

# NOT

Bit inversion

**NOT** performs a bit inversion of input variables. The result will be written into the output variable.




## Parameters

Input

**Unnamed input** (BOOL, WORD, DWORD)

Input for NOT operation

Output

**Unnamed output** (BOOL, WORD, DWORD)

Output as input: Result

## Remarks

All operands must be of the same data type.

# Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

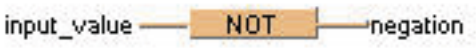
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | WORD | 0 | type: BOOL, WORD or DWORD |
| 1 | VAR | negation | WORD | 0 | type: BOOL, WORD or DWORD |

This example uses variables. You can also use a constant for the input variable.

## POU body

The bits of **input_value** are inversed (0 is inversed to 1 and vice versa). The inversed result is written into **negation**.

### LD body



## 5.8 FP instructions

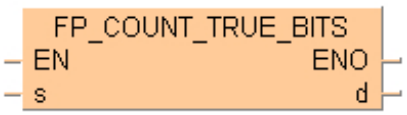
Tip

Advantages of FP instructions

# FP_COUNT_TRUE_BITS

Number of ON bits

This FP instruction counts the number of bits which are TRUE in the data specified by **s** if the trigger **EN** is TRUE. The result is stored in **d**.




## Parameters

Input

**s** (WORD, DWORD)

Input value

Output

**d** (INT, DINT, UINT, UDINT)

Number of bits which are TRUE in the argument applied at **s**

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

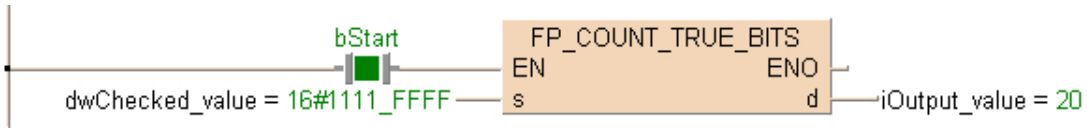
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | dwChecked_value | DWORD | 16#1111FFFF | this value will be checked |
| 2 | VAR | iOutput_value | INT | 0 | result after a 0->1 leading |

## POU body

When the variable **bStart** is set to TRUE, the function is carried out. The number of bits which are TRUE in the binary equivalent of 16#1111FFFF is 20.

| 31 ... 28 | 27 ... 24 | 23 ... 20 | 19 ... 16 | 15 ... 12 | 11 ... 8 | 7 ... 4 | 3 ... 0 |
|---|---|---|---|---|---|---|---|
| 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 0 0 0 1 | 1 1 1 1 | 1 1 1 1 | 1 1 1 1 | 1 1 1 1 |

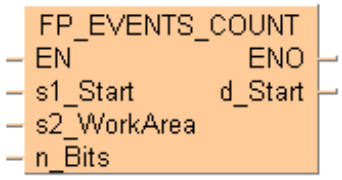
### LD body


bStart
FP_COUNT_TRUE_BITS
EN
ENO
dwChecked_value = 16#1111_FFFF
s
d
iOutput_value = 20

# FP_EVENTS_COUNT

Counts the number of rising edges for each bit

This FP instruction counts the number of rising edges of each bit in the data area specified by the starting address **s1_Start** and the number of bits **n_Bits**. The counting results are stored in **n_Bits** elements of data type DINT or UDINT starting with **d_Start**.




## Parameters

Input

**s1_Start** (WORD, INT, UINT)

Starting address of the data area for the bits to be evaluated.
Use the following formula to determine the minimum size of this data area:

$$\text{size} = \frac{(\text{n Bits} + 15)}{16}$$

All decimal places in the result of this integer division must be removed.

- **n_Bits** from 1–16 => size is 1 word.
- **n_Bits** from 17–32 => size is 2 words...

**s2_WorkArea** (WORD, INT, UINT)

Starting address of the working area. Its size must be identical to the size of **s1_Start**.

**n_Bits** (INT, DINT, UINT, UDINT)

Number of bits (permissible range: 1–65535)

Output

**d_Start** (WORD, INT, UINT)

Starting address of the data area storing the counting results consisting of minimum **n_Bits** elements of data type DINT or UDINT. Please use the instructions **Adr_Of_Var_I** (LD/FBD) or **Adr_Of_Var** (ST) to apply this data area.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

- if **d_Start**, **s1_Start**, or **s2_WorkArea** are out of range.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **d_Start**, **s1_Start**, or **s2_WorkArea** are out of range.

## Example

### POU header

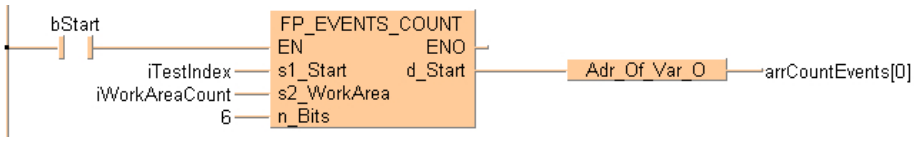
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | iTestIndex | INT | 0 |
| 2 | VAR | iWorkAreaCount | INT | 0 |
| 3 | VAR | arrCountEvents | ARRAY [0..5] of dint | [6(0)] |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

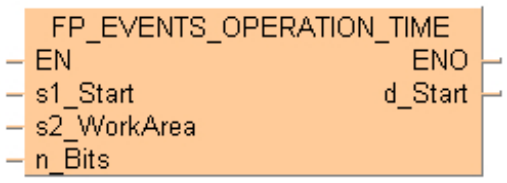
### LD body



# FP_EVENTS_OPERATION_TIME

Counts the operation time of each bit

This FP instruction measures the duration of the TRUE state of each bit in the data area specified by the starting address **s1_Start** and the number of bits **n_Bits**. The results are stored in the area of **n_Bits***2 words starting with **d_Start**. The operation time is indicated in seconds.




## Parameters

Input

**s1_Start** (WORD, INT, UINT)

Starting address of the data area for the bits to be evaluated.
Use the following formula to determine the size of the data type:

$$\text{size} = \frac{(\text{n Bits} + 15)}{16}$$

All decimal places in the result of this integer division must be removed.

- **n_Bits** from 1 to 16=> size is 1 word.
- **n_Bits** from 17 to 32=> size is 2 words...

**s2_WorkArea** (WORD, INT, UINT)

Starting address of the working area. Its size must be identical to the size of **s1_Start**.

**n_Bits** (INT, DINT, UINT, UDINT)

Number of bits (permissible range: 1–65535)

Output

**d_Start** (WORD, INT, UINT)

Starting address of the data area storing the counting results consisting of minimum **n_Bits** elements of data type DINT or UDINT. Please use the instructions **Adr_Of_Var_I** (LD/FBD) or **Adr_Of_Var** (ST) to apply this data area.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **d_Start**, **s1_Start**, or **s2_WorkArea** are out of range.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **d_Start**, **s1_Start**, or **s2_WorkArea** are out of range.

## Example

### POU header

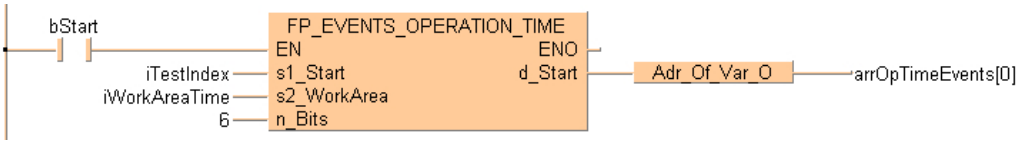
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | iTestIndex | INT | 0 |
| 2 | VAR | iWorkAreaTime | INT | 0 |
| 3 | VAR | arrOpTimeEvents | ARRAY [0..5] of dint | [6(0)] |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

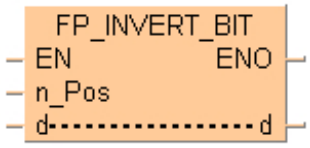
### LD body



# FP_INVERT_BIT

Bit invert

This FP instruction inverts [1 (TRUE) →0 (FALSE) or 0 (FALSE) →1 (TRUE)] the bit at bit position **n_Pos** in the data area specified by **d** if the trigger **EN** is TRUE. Bits other than the bit specified do not change.




## Parameters

Input

**n_Pos** (INT, DINT, UINT, UDINT)

Bit position to be inverted (permissible range: 0–15)

Input/output

**d** (WORD)

Result

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## POU header

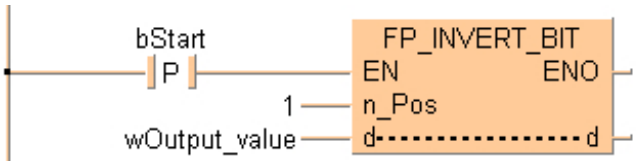
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 1 | VAR | bStart | BOOL | FALSE | activates the function |
| 2 | VAR | wOutput_value | WORD | 2#111 | result after a 0->1 leading edge from start: 2#101 |
| 3 | VAR | | | | |

## POU body

When the variable **bStart** is set to TRUE, the function is carried out.

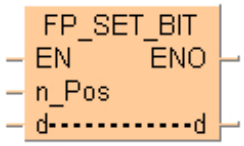
## LD body



# FP_SET_BIT

Bit set

This FP instruction turns ON the bit specified by the bit position at **n_Pos** of the data specified by **d** if the trigger **EN** is in the ON-state. Bits other than the bit specified do not change.




## Parameters

Input

**n_Pos** (INT)

Bit position to be set (permissible range: 0–15)

Input/output

**d** (WORD)

16-bit data area

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## POU header

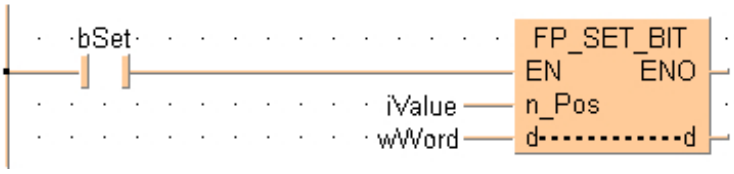
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bSet | BOOL | FALSE | activates the function |
| 1 | VAR | iValue | INT | 0 | bit position |
| 2 | VAR | wWord | WORD | 2#101010 | result after a 0->1 leading edge from start: 2#101011 |
| 3 | VAR | | | | |

## POU body

When the variable **bSet** is set to TRUE, the function is carried out. The bit on position 0 is set to 1.

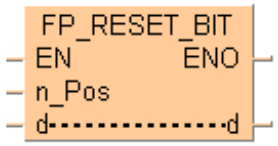
## LD body



# FP_RESET_BIT

Bit reset

This FP instruction turns OFF the bit specified by the bit position at **n_Pos** of the data specified by **d** if the trigger **EN** is in the ON-state. Bits other than the bit specified do not change. The range of **n_Pos** is 0–15.




## Parameters

Input

**n_Pos** (INT)

Bit position to be reset (permissible range: 0–15)

Input/output

**d** (WORD)

16-bit data area

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## POU header

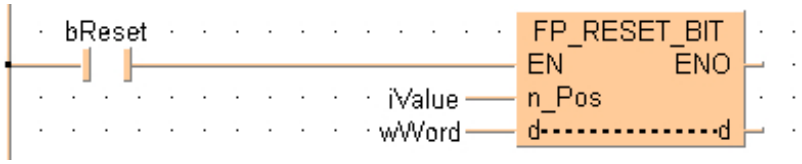
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 1 | VAR | bReset | BOOL | FALSE | activates the function |
| 2 | VAR | iValue | INT | 2 | bit position |
| 3 | VAR | wWord | WORD | 2#10101 | result after a 0->1 leading edge from start: 2#10001 |
| 4 | VAR | | | | |

## POU body

When the variable **bReset** is set to TRUE, the function is carried out. The bit on position 2 is reset to 0.

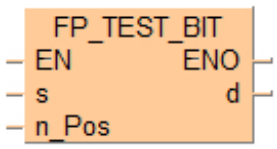
## LD body



# FP_TEST_BIT

Bit test

This FP instruction checks the state 1 (TRUE) or 0 (FALSE) of bit position **n_Pos** in the input value at **s** if the trigger **EN** is in the ON-state. The result is stored in **d**.




## Parameters

Input

**s** (WORD)

Input value

**n_Pos** (INT, DINT, UINT, UDINT)

Specifies the bit position to be checked in decimal data (permissible range: 0–15)

Output

**d** (BOOL)

State of bit position

## Remarks

The specified bit is checked by the system variable **sys_bIsEqual**.

- When specified bit is 0 (OFF), **sys_bIsEqual** (=flag) turns ON.
- When specified bit is 1 (ON), **sys_bIsEqual** (=flag) turns OFF.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

### POU header

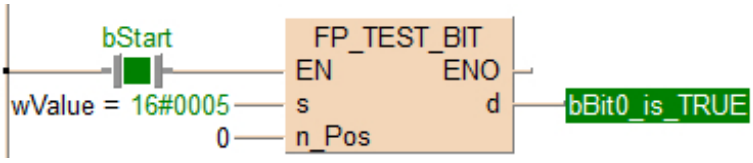
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | bBit0_is_TRUE | BOOL | FALSE | TRUE if bit LSB of input value is TRUE else FALSE |
| 2 | VAR | wValue | WORD | 2#101 | input value |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

### LD body



## 5.9 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

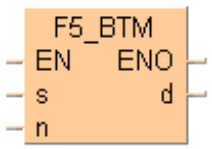
Advantages of IEC instructions

Advantages of FP instructions

# F5_BTM

Bit data move

1 bit of the 16-bit data or constant value specified by **s** is copied to a bit of the 16-bit area specified by **d** according to the content specified by **n** if the trigger **EN** is in the ON-state. When the 16-bit equivalent constant is specified by **s**, the bit data move operation is performed internally converting it to 16-bit binary expression.




## Parameters

Input

**s** (WORD, INT, UINT)

Source 16-bit area

**n** (WORD, INT, UINT)

Specifies source and destination bit positions

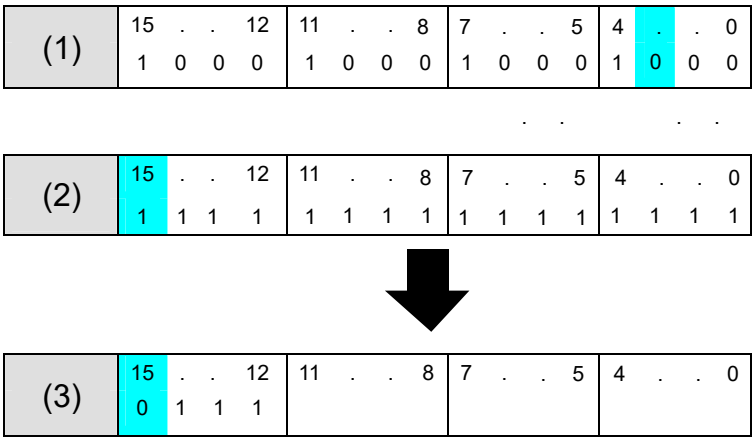
Output

**d** (WORD, INT, UINT)

Destination 16-bit area

### Example

Explanation with source value **s**=16#8888, source bit position 02 and destination bit position 0F defined by **n**=16#0F02




(1) Source value 16#8888 with source bit position 02
(2) Target value 16#FFFF with target bit position 15
(3) Result value 16#7FFF with bit 02 moved to bit 15

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_MOVE_BITS** (page 926)
- The variables **s** and **d** have to be of the same data type.
- The operand **n** specifies the bit number as follows:

| | | | | | |
|---|---|---|---|---|---|
| 16# | 0 | F | 0 | 2 | |
| | | ↑ | | ↑ | Source bit position |
| | | Destination bit position | | | |

| | |
|---|---|
| Bit No. 0 to 3 | Source bit No. (16#0 to 16#F) |
| Bit No. 4 to 7 | FP2/2SH and 10SH: number of bits to be transferred (16#0 to 16#F)<br>FP3: invalid |
| Bit No. 8 to 11 | Destination bit No. (16#0 to 16#F) |
| Bit No. 12 to 15 | Invalid |

For example, reading from the right, **n** = 16#C01 would move from bit position one, one bit to bit position 12 (16#C).

## Example

## POU header

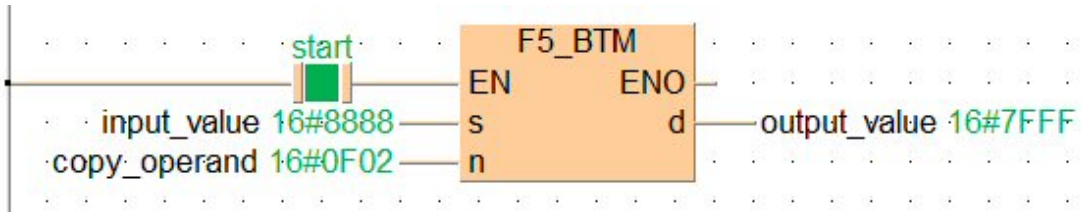
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | input_value | WORD | 2#1000100010001000 | |
| 2 | VAR | copy_operand | WORD | 16#0F02 | digit no.1 and no.3 are invalid, digit no.0 locates |
| 3 | VAR | output_value | WORD | 2#1111111111111111 | result after a 0->1 leading edge from start: 2#0111111111111111 |
| 4 | VAR | | | | |

## POU body

When the variable **start** is set to TRUE, the function is carried out.

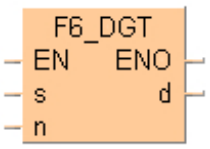
## LD body



# F6_DGT

Digit data move

The hexadecimal digits in the 16-bit data or in the 16-bit equivalent constant specified by **s** are copied to the 16-bit area specified by **d** as specified by **n**.




## Parameters

Input

**s** (WORD, INT, UINT)

Source 16-bit area

**n** (WORD, INT, UINT)

Specifies source and destination hexadecimal digit position and number of hexadecimal digits

Output

**d** (WORD, INT, UINT)

Destination 16-bit area

## Example

## POU header

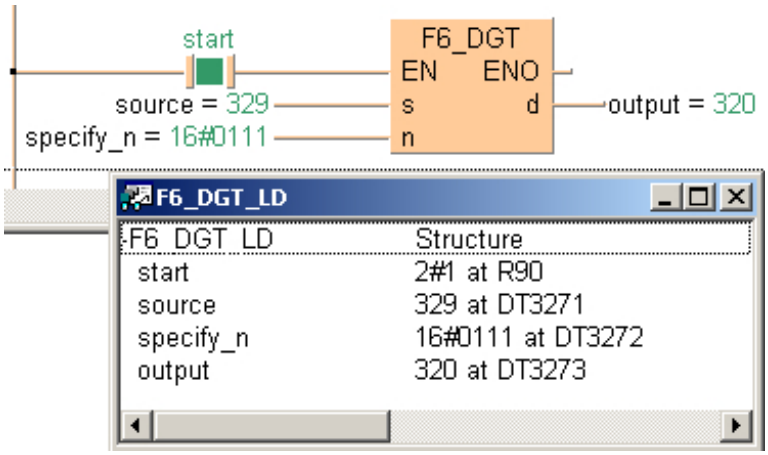
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | |
| 1 | VAR | source | INT | 329 | decimal 329 = 16#149 |
| 2 | VAR | specify_n | WORD | 16#111 | Beginning from the end: |
| 3 | VAR | output | INT | 0 | 1: first hex. digit is digit 1, i.e. 4 |
| 4 | VAR | | | | 1: copies 2 hex. digits, i.e. 14 |
| | | | | | 1: destination is hex. digit 1 |

### POU body

When the variable **start** is set to TRUE, the function is carried out. The values for **source** and **output** in the “Monitor header” of the ladder diagram body have been set to display the hexadecimal value by activating the “Hex” button in the toolbar.

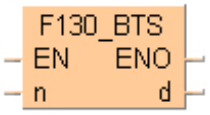
### LD body



# F130_BTS

16-bit data bit set

Turns ON the bit specified by the bit position at **n** of the 16-bit data specified by **d** if the trigger **EN** is in the ON-state. Bits other than the bit specified do not change. The range of **n** is 0 to 15.




## Parameters

Input

**d** (WORD, INT, UINT)

16-bit area

Output

**n** (INT)

Specifies bit position to be set

## Example

## POU header

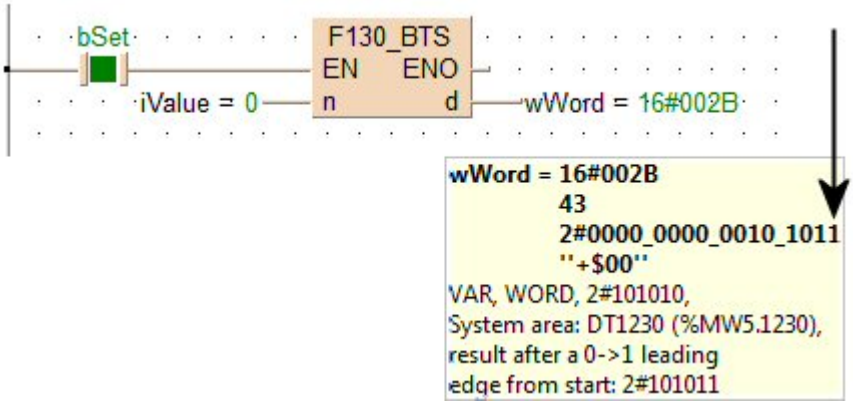
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bSet | BOOL | FALSE | activates the function |
| 1 | VAR | iValue | INT | 0 | bit position |
| 2 | VAR | wWord | WORD | 2#101010 | result after a 0->1 leading edge from start: 2#101011 |
| 3 | VAR | | | | |

## POU body

When the variable **bSet** is set to TRUE, the function is carried out. The bit on position 0 is set to 1.

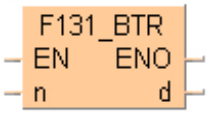
### LD body


bSet
F130_BTS
EN
ENO
iValue = 0
n
d
wWord = 16#002B
wWord = 16#002B
43
2#0000_0000_0010_1011
''+$00''
VAR, WORD, 2#101010,
System area: DT1230 (%MW5.1230),
result after a 0->1 leading
edge from start: 2#101011

# F131_BTR

16-bit data bit reset

Turns OFF the bit specified by the bit position at **n** of the 16-bit data specified by **d** if the trigger **EN** is in the ON-state. Bits other than the bit specified do not change. The range of **n** is 0 to 15.




## Parameters

Input

**n** (INT)

Specifies bit position to be reset

Output

**d** (WORD, INT, UINT)

16-bit area

## Example

## POU header

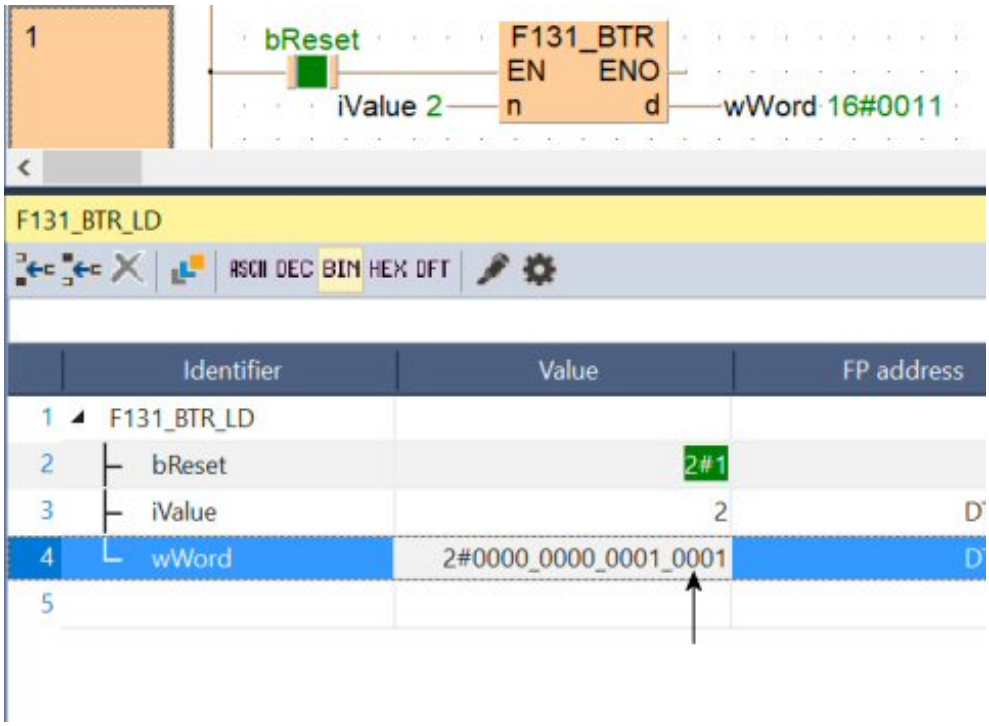
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 1 | VAR | bReset | BOOL | FALSE | activates the function |
| 2 | VAR | iValue | INT | 2 | bit position |
| 3 | VAR | wWord | WORD | 2#10101 | result after a 0->1 leading edge from start: 2#10001 |
| 4 | VAR | | | | |

## POU body

When the variable **bReset** is set to TRUE, the function is carried out. The bit on position 2 is reset to 0.

**LD body**

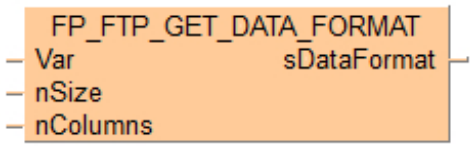

1
bReset
F131_BTR
EN
ENO
iValue 2
n
d
wWord 16#0011
F131_BTR_LD
ASCII DEC BIN HEX OFT
Identifier
Value
FP address
1 F131_BTR_LD
2 bReset
2#1
3 iValue
2
4 wWord
2#0000_0000_0001_0001
5

# 6 Communication instructions for Ethernet clients

# FP_FTP_GET_DATA_FORMAT

Generate data format string for FTP protocol

This FP instruction generates a string specifying the data format (FP address and length) according to the data type at the input **Var**. It writes the string into the output variable **sDataFormat** that is suitable for the input **sPLCFullFileNameOrDataFormat** of the instruction **FP_FTP_SET_MODE**.




## Parameters

Input

**Var** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Simple data type or array of simple data types, e.g. BOOL, INT, UINT, STRING, ...
Restriction: ARRAY..OF..STRING is not allowed

**nSize** (WORD, INT, UINT)

Size of data in word or for strings in byte units.

**nColumns** (WORD, INT, UINT)

Number of items after which a new line should be added in the .csv converted file

Output

**sDataFormat** (STRING)

String specifying the data format for the input **sPLCFullFileNameOrDataFormat** of **FP_FTP_SET_MODE**

## Remarks

- Before you execute the instruction, you need to specify the transfer settings using **FP_FTP_SET_MODE** or the setting dialog of the FTP client.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- The instruction can only be executed when the transfer request flag for the specified transfer setting is "FALSE: No transfer request". When the transfer request flag is "TRUE: Transfer requested", an operation error occurs.

- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- This instruction does not overwrite the Ethernet configuration data stored in the PLC permanently. When the PLC has been switched off and on again, the Ethernet configuration data stored in the PLC are used again.
- After the FTP client transfer settings have been completed, data is sent to files or obtained from files when **FP_FTP_TRANSFER_REQUEST** is executed.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a timeout of the connection is exceeded
- if an IP address is invalid

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a timeout of the connection is exceeded
- if an IP address is invalid

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetModePlcData | BOOL | FALSE |
| 1 | VAR | bSetModePlcDataError | BOOL | FALSE |
| 2 | VAR | arrayValues | ARRAY[0..15] OF REAL | [16(0.0)] |
| 3 | VAR | iID10 | INT | 10 |

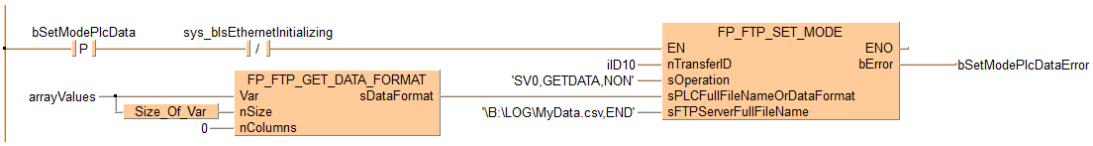
### POU body

If **bSetModePlcData** changes from FALSE to TRUE and **sys_bIsEthernetInitializing** is FALSE, the instruction is carried out.

## LD body

Note

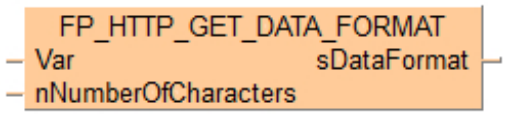
When a constant with the value 0 is applied to the input **nColumns**, the line feed position in the generated data format string is omitted. This is necessary if the instruction **FP_FTP_SET_MODE** is used subsequently in the operation mode GETDATA.



# FP_HTTP_GET_DATA_FORMAT

Generate data format string for HTTP protocol

This FP instruction generates a string specifying the data format (FP address and length). It writes the string into the output variable **sDataFormat** that is suitable for the input **sPLCDataFormat** of the instruction **FP_HTTP_SET_MODE**.




## Parameters

Input

**Var** (STRING)

String containing ASCII data

**nNumberOfCharacters** (WORD, INT, UINT)

Size of data in word or for strings in byte units.

Output

**sDataFormat** (STRING)

String specifying the data format for the input **sPLCDataFormat** of **FP_HTTP_SET_MODE**

## Remarks

- Before you execute the instruction, you need to specify the transfer settings using **FP_HTTP_SET_MODE** or the setting dialog of the HTTP client.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- The instruction can only be executed when the transfer request flag for the specified transfer setting is "FALSE: No transfer request". When the transfer request flag is "TRUE: Transfer requested", an operation error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- This instruction does not overwrite the Ethernet configuration data stored in the PLC permanently. When the PLC has been switched off and on again, the Ethernet configuration data stored in the PLC are used again.

- After the HTTP client transfer settings have been configured, data is actually sent or acquired when the instruction **FP_HTTP_TRANSFER_REQUEST** is executed.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a timeout of the connection is exceeded
- if an IP address is invalid

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a timeout of the connection is exceeded
- if an IP address is invalid

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

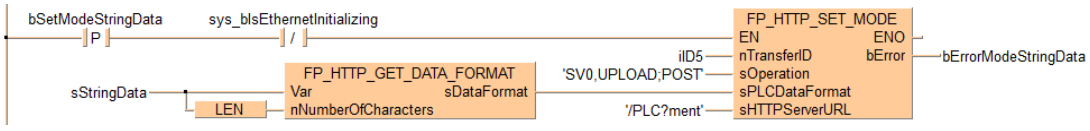
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | sStringData | STRING[31767] | '' |
| 1 | VAR | iID5 | INT | 5 |
| 2 | VAR | bSetModeStringData | BOOL | FALSE |
| 3 | VAR | bErrorModeStringData | BOOL | FALSE |

### POU body

If **bSetModeStringData** changes from FALSE to TRUE and **sys_bIsEthernetInitializing** is FALSE, the instruction is carried out.

### LD body

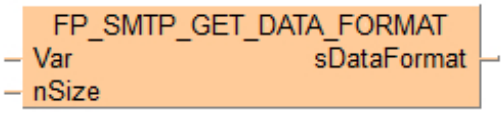


# FP_SMTP_GET_DATA_FORMAT

Generate data format string for SMTP protocol

This FP instruction generates a string specifying the data format (FP address and length) according to the data type at the input **Var**. It writes the string into the output variable **sDataFormat** that is suitable for the input **sAttachment** of the instruction **FP_SMTP_SET_MODE**.

When data registers should be transmitted with **FP_SMTP_SET_MODE**, a .CSV file of the given data values is generated. During transfer, the data is converted to ASCII according to the data type. Therefore, the **sAttachment** input needs additional information indicating which data should be converted to which ASCII type.




## Parameters

Input

**Var** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Simple data type or array of simple data types e.g. BOOL, INT, UINT, STRING, ...
Restriction: ARRAY..OF..STRING is not allowed

**nSize** (WORD, INT, UINT)

Size of data in word or for strings in byte units.

Output

**sDataFormat** (STRING)

String specifying the data format for the input **sAttachment** of **FP_SMTP_SET_MODE**

## Remarks

- Before you execute the instruction, you need to specify the e-mail transmission settings using **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client.
- Before you execute the instruction, you need to specify the group and event e-mail settings using **FP_SMTP_SET_GROUP** or the setting dialog of the SMTP client.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.

- The instruction can only be executed when the transfer request flag for the specified transfer setting is "FALSE: No transfer request". When the transfer request flag is "TRUE: Transfer requested", an operation error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a timeout of the connection is exceeded
- if an IP address is invalid

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a timeout of the connection is exceeded
- if an IP address is invalid

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

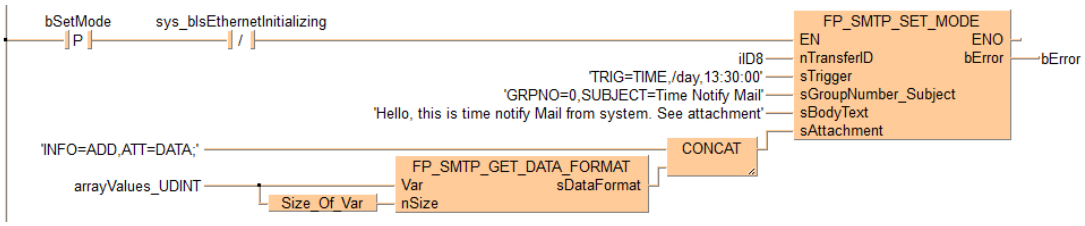
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | arrayValues_UDINT | ARRAY [0..19] OF UDINT | |
| 1 | VAR | bError | BOOL | FALSE |
| 2 | VAR | bSetMode | BOOL | FALSE |
| 3 | VAR | iID8 | INT | 8 |

### POU body

If **bSetMode** changes from FALSE to TRUE and **sys_bIsEthernetInitializing** is FALSE, the instruction is carried out.

## LD body


bSetMode
P
sys_blsEthernetInitializing
/
FP_SMTP_SET_MODE
EN
ENO
nTransferID
bError
bError
iID8
'TRIG=TIME,/day,13:30:00'
sTrigger
'GRPNO=0,SUBJECT=Time Notify Mail'
sGroupNumber_Subject
'Hello, this is time notify Mail from system. See attachment'
sBodyText
sAttachment
'INFO=ADD,ATT=DATA;'
CONCAT
FP_SMTP_GET_DATA_FORMAT
arrayValues_UDINT
Var
sDataFormat
Size_Of_Var
nSize

## 6.4 FP instructions

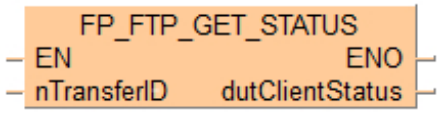
Tip

Advantages of FP instructions

# FP_FTP_GET_STATUS

Get status of a single Ethernet unit using FTP transmission

This FP instruction gets the information from the Ethernet unit specified by **nTransferID** and writes the values into the DUT **FP_CLIENT_STATUS_DUT**.




## Parameters

Input

**nTransferID** (WORD, INT, UINT)

Ethernet unit ID (values: 0–15)

Output

**dutClientStatus** (**FP_CLIENT_STATUS_DUT**)

Stores the values of the Ethernet status of the specified Ethernet unit

## Remarks

- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the transfer settings using **FP_FTP_SET_MODE** or the setting dialog of the FTP client.
- Before you execute the instruction, you need to specify the transfer settings for data recording files using **FP_FTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the FTP client.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the Ethernet unit ID specified is outside the permissible range (0–15)
- if a transfer setting that has not been configured with **FP_FTP_SET_MODE** or in the setting dialog of the FTP client is specified.
- if a transfer setting for data recording files that has not been configured with the instruction **FP_FTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the FTP client is specified.
- if the instruction is executed in an interrupt program

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the Ethernet unit ID specified is outside the permissible range (0–15)
- if a transfer setting that has not been configured with **FP_FTP_SET_MODE** or in the setting dialog of the FTP client is specified.
- if a transfer setting for data recording files that has not been configured with the instruction **FP_FTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the FTP client is specified.
- if the instruction is executed in an interrupt program

## Example

### POU header

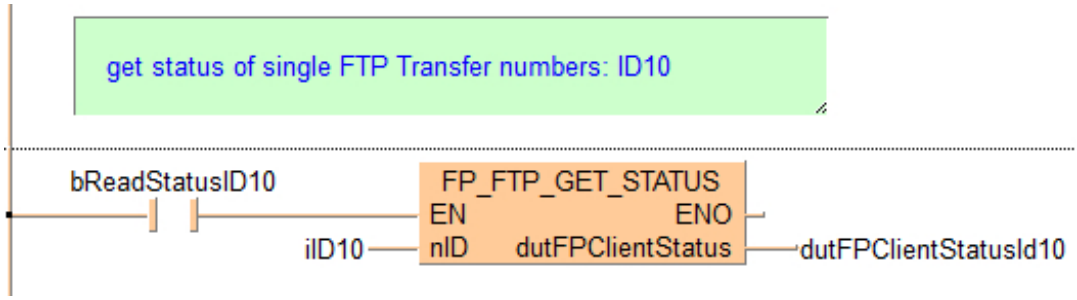
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bReadStatusID10 | BOOL | FALSE |
| 1 | VAR | dutFPClientStatusId10 | FP_CLIENT_STATUS_DUT | |
| 2 | VAR | iID10 | INT | 10 |

### POU body

If **bReadStatusID10** is set to TRUE, the instruction is carried out. It gets the FTP transfer values from Ethernet unit 10 and writes the values into the DUT

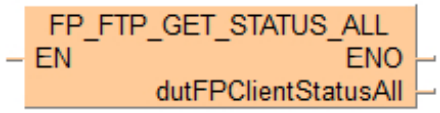
### LD body



# FP_FTP_GET_STATUS_ALL

Get status of all Ethernet units using FTP transmission

This FP instruction gets the information from all Ethernet units **nTransferID0**–**nTransferID15** and writes the values into the DUT **FP_CLIENT_STATUS_ALL_DUT**.




## Parameters

Output

**dutFPClientStatusAll** (**FP_CLIENT_STATUS_ALL_DUT**)

Stores the values of the Ethernet status of all Ethernet units

## Remarks

- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the transfer settings using **FP_FTP_SET_MODE** or the setting dialog of the FTP client.
- Before you execute the instruction, you need to specify the transfer settings for data recording files using **FP_FTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the FTP client.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a transfer setting that has not been configured with **FP_FTP_SET_MODE** or in the setting dialog of the FTP client is specified.
- if a transfer setting for data recording files that has not been configured with the instruction **FP_FTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the FTP client is specified.
- if the instruction is executed in an interrupt program

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a transfer setting that has not been configured with **FP_FTP_SET_MODE** or in the setting dialog of the FTP client is specified.

- if a transfer setting for data recording files that has not been configured with the instruction **FP_FTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the FTP client is specified.
- if the instruction is executed in an interrupt program

## Example

### POU header

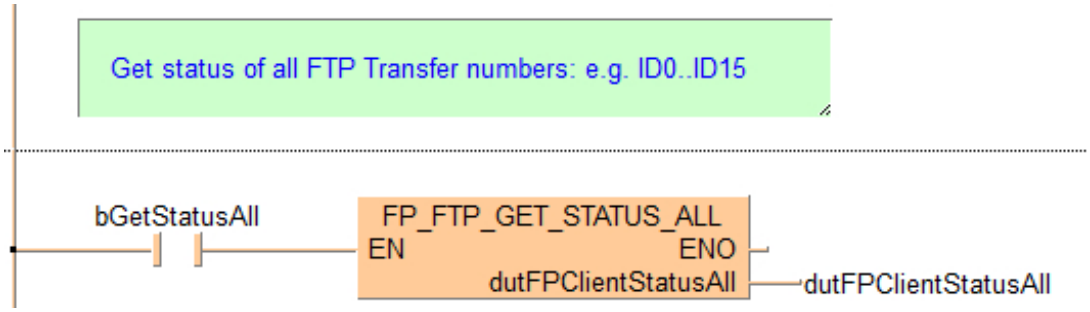
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | dutFPClientStatusAll | FP_CLIENT_STATUS_ALL_DUT | |
| 1 | VAR | bGetStatusAll | BOOL | FALSE |

### POU body

If **bGetStatusAll** is set to TRUE, the instruction is carried out. The status values of all Ethernet units are written into the DUT **dutFPClientStatusAll**.

### LD body



# FP_FTP_SET_CONNECTION

Server settings for FTP connection

This FP instruction defines the server settings for the connection to the FTP client specified in the CPU according to specified parameters.


## Parameters

Input

**sServer** (STRING)

- Server address (required parameter)
  Keyword: `SV`
  Values: SV0–SV3 (Server 0–Server 3)
- IP address or host name (required parameter).
  - IP address
    For an IP address, specify the keyword `IPv4` or `IPv6` at the beginning.
    Syntax for IPv4: e.g. `'IPv4=111.122.133.144'`
    Syntax for IPv6: e.g. `'IPv6=1111:122:2:1555:0:0:1888'`
    Please note that for IPv4 addresses there are range restrictions. When an invalid IP address is specified with an instruction, there will be no operation error but the output **bError** will be set to TRUE.
  - Host name
    Keyword: `HOST`
    Syntax: e.g.`'HOST=FTP.pidsx.com'`
- Port number (optional parameter)
  Keyword: `'PORT'`
  Syntax: `'PORT=xxxxx'`
  Values: 1–65535 (default: 21)
- Open method (optional parameter)
  Keyword: `OPEN`
  Syntax: `'OPEN=xxxx'`
  Values: `act` (active), `pasv` (passive) (default: `act`)
- SSL3/TSL1 authentication (optional parameter)
  Specify whether or not to use SSL3/TSL1 authentication.

Keywords:

- `SSL`: Use SSL3/TLS1
- `NON`: SSL3/TLS1 not used (default)

Examples:

1. Connect to FTP server number 0 with the IP address 192.255.2.10, port number 21, active open method, using SSL3/TLS1 authentication: `'SV0,IPv4=192.255.2.10,PORT=21,OPEN=act,SSL'`
2. Connect to FTP server number 1 with the IP address 1111:1222::1555:0:0:1888, port number omitted (use default port 21), open method omitted (use default = active) using SSL3/TLS1 authentication: `'SV1,IPv6=1111:1222::1555:0:0:1888,SSL'`
3. Connect to FTP server number 2 with the host name FTP.pidsx.com, port number 28, passive open method, do not use authentication: `'SV2,HOST=FTP.pidsx.com,PORT=28,OPEN=pasv,NON'`

**sLogin** (STRING)

Set the login data

- User name (max. 32 characters)

  Keyword: `USER`

  Syntax: `'USER=xxx'` (default: root). Use `'USER='` to delete the user name.

- Password (max. 32 characters, upper and lower case characters are allowed)

  Keyword: `PASS`

  Syntax: `'PASS=passwd'` (default: root). Use `'PASS='` to delete the password.

For this parameter, there are two additional keywords available:

- `INITIAL`: Resets user name and password to the default settings "root" and "root".
- `KEEP`: Keeps the current login settings.

Examples:

1. Set user name to "Admin" and password to "Panasonic": `'USER=Admin,PASS=Panasonic'`
2. Set user name to "Supervisor" and delete the password: `'USER=Supervisor,PASS='`
3. Delete user name and password: `'USER=,PASS='`
4. Reset user name and password to the default values: `'INITIAL'`
5. Keep the current user name and password: `'KEEP'`

**sCommunication** (STRING)

Set optional communication parameters as required.

- Connection timeout

  Keyword: `TOUT`

  Syntax: `'TOUT=xxx'` (default: 60 seconds)

  Values: 30–300 seconds

- Number of retries

  Keyword: `RTRY`

Syntax: `'RTRY=x'` (default: 3 times)

Values: 0–3

- Retry interval

  Keyword: `RTTM`

  Syntax: `'RTTM=xxxxx'` (default: 600 seconds)

  Values: 10–86400 seconds

  The value can be specified by 10 seconds. It is rounded down to the 10.

  Example: When you specify 38 seconds, 30 seconds are set.

For this parameter, there are two additional keywords available:

- `INITIAL`: Resets connection timeout, number of retries, and the retry interval to the default settings.
- `KEEP`: Keeps the current communication settings.

Examples:

1. Set connection timeout: 30 seconds, number of retries: 2, retry interval: 500 seconds: `'TOUT=30,RTRY=2,RTTM=500'`
2. Set connection timeout: 270 seconds, no retries, retry interval: 4900 seconds: `'TOUT=270,RTRY=0,RTTM=4900'`
3. Set connection timeout: 30 seconds, number of retries: 25, retry interval: do not change: `'TOUT=30,RTRY=25'`
4. Set connection timeout: do not change, number of retries: 25, retry interval: 3000 seconds: `',RTRY=25,RTTM=3000'`
5. Reset to default settings (connection timeout: 60 seconds, number of retries: 3, retry interval: 600 seconds): `'INITIAL'`
6. Keep all the current settings: `'KEEP'`

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- Separate all keyword entries by comma. e.g. 'NAME=abcd,FROM=sender@server.com'
- The number of characters for string data must not exceed 256.
- This instruction is not available in interrupt programs.
- Upper and lower case characters can be used for operands for which a character constant can be specified. "Abcd", "ABCD" and "abcd" are synonymous, however, file names are case-sensitive.

- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- The instruction can only be executed when the transfer request flag for the specified transfer setting or the specified **nLogID** number is FALSE. When the transfer request flag is TRUE, an operation error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.
- if the same keyword is specified more than once
- if the transfer request flag for the specified transfer setting is "TRUE: Transfer requested" when the instruction is executed.
- if the transfer request flag for a specified **nLogID** number is TRUE, e.g. if **sys_bIsLog0DataRecordingActive** is TRUE.
- if server numbers are not specified in ascending order.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.
- if the same keyword is specified more than once
- if the transfer request flag for the specified transfer setting is "TRUE: Transfer requested" when the instruction is executed.
- if the transfer request flag for a specified **nLogID** number is TRUE, e.g. if **sys_bIsLog0DataRecordingActive** is TRUE.
- if server numbers are not specified in ascending order.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed with an incorrect IP address, **sys_iEthernetConnectionErrorCode** is set to "1: Incorrect IP address is specified"
- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

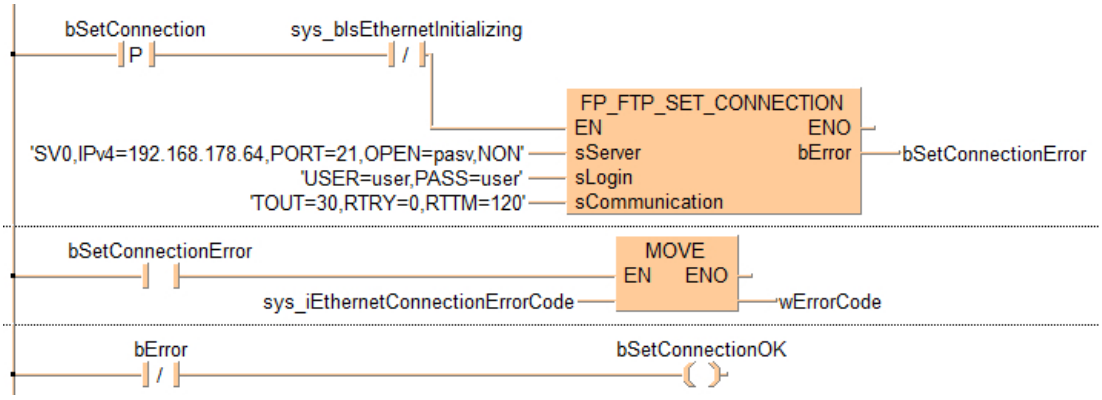
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bError | BOOL | FALSE |
| 1 | VAR | wErrorCode | WORD | 0 |
| 2 | VAR | bSetConnectionOK | BOOL | FALSE |
| 3 | VAR | bSetConnectionError | BOOL | FALSE |
| 4 | VAR | bSetConnection | BOOL | FALSE |

### POU body

If **bSetConnection** changes from FALSE to TRUE and **sys_bIsEthernetInitializing** is FALSE, the instruction is carried out.

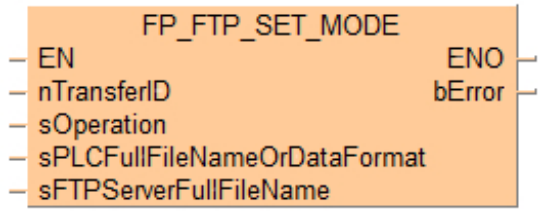
### LD body



# FP_FTP_SET_MODE

Set FTP transfer mode

This FP instruction stores the FTP client transfer settings of **sOperation**, **sPLCFullFileNameOrDataFormat** and **sFTPServerFullFileName** in the transfer setting area specified by **nTransferID**.




## Parameters

Input

**nTransferID** (WORD, INT, UINT)

Set the transfer setting ID.
Values: 0–15 (values must be entered in ascending order)

**sOperation** (STRING)

Set the transfer method parameters.

- Server address (required parameter)

  Keyword: `SV`

  Syntax: `'SVx'`

  Values: SV0–SV3 (Server 0–Server 3)

- Transfer target and method (required parameter)

| Parameter string | Target | Transfer method |
|---|---|---|
| `PUTFILE` | File | Send to servers (Overwrite method) |
| `PUTFILE-OVW` | | Send to servers (Overwrite method) |
| `PUTFILE-REN` | | Send to servers (Rename method) |
| `GETFILE` | | Obtain from servers |
| `PUTDATA` | Memory area | Send to servers (Overwrite method) |
| `PUTDATA-OVW` | | Send to servers (Overwrite method) |
| `PUTDATA-REN` | | Send to servers (Rename method) |
| `GETDATA` | | Obtain from servers |

- File handling after transfer (required parameter)
  - `DEL`: delete the file

– NON: do not delete the file

Examples:

1. Send file to FTP server 1 using overwrite method, do not delete file after transfer: 'SV1,PUTFILE-OVW,NON'
2. Send file to FTP server 0 using rename method, delete file after transfer: 'SV0,PUTFILE-REN,DEL'
3. Get file from FTP server 2, delete file after transfer: 'SV2,GETFILE,DEL'

**sPLCFullFileNameOrDataFormat** (STRING)

Make the settings according to your transfer target (file or data from memory area) and the transfer method (PUT or GET).

Transfering data of a variable can be done by using the instruction **FP_FTP_GET_DATA_FORMAT**. The output of that instruction will be connected to **sPLCFullFileNameOrDataFormat** to read the string specifying the data format string of the variable (memory area, size, and data type). The transmitted data is written to or read from a CSV file according to the data type of the variable.

- Transfer target: file

  Set the source file name.

  – For PUTFILE: Specify the file name of a file stored on an SD card with a relative path. Example: '\LOG\MyData.csv'

  – For GETFILE: Specify a file name from the home directory of a user which logs in FTP servers with a relative path. Example: '\LOG\MyData.csv'

- Transfer target: data from memory area

  – For PUTDATA: The data stored in the variable connected to input of **FP_FTP_GET_DATA_FORMAT** is written into the CSV file specified by **sFTPServerFullFileName**. For this transfer method, it is possible to add a time stamp in the format yymmdd_hhmmss to the file name. Use the keywords 'TOP' and 'END' to specify whether to add the time stamp **before** or **after** the file name.

  – For GETDATA: The data of the CSV file specified by **sFTPServerFullFileName** is written into the variable connected to the input **Var** of **FP_FTP_GET_DATA_FORMAT**.

**sFTPServerFullFileName** (STRING)

Make the settings according to your transfer target (file or data from memory area) and the transfer method (PUT or GET).

- Transfer target: file

  – For PUTFILE: Specify a folder name with its relative path from the home directory of the user who logs in to the FTP server. For specifying the home directory, specify "/" or “\" only.

  – For GETFILE: Specify a storage folder name in an SD card with an relative path.

- Transfer target: data from memory area

  – For PUTDATA: Specify a destination CSV file name to write data into. For this transfer method, it is possible to add a time stamp in the format yymmdd_hhmmss

to the file name. Use the keywords `'TOP'` and `'END'` to specify whether to add the time stamp **before** or **after** the file name.

Examples:

1. Transfer data to the destination file "PutData1.bin" located in the directory "\FTP", do not add a time stamp: `'\FTP\PutData1.bin'`
2. Transfer data to the destination file "PutData2.bin" located in the directory "\FTP", add a time stamp at the beginning of the file name: `'\FTP\PutData2.bin,TOP'`
3. Transfer data to the destination file "PutData3.bin" located in the directory "\FTP", add a time stamp at the end of the file name: `'\FTP\PutData3.bin,END'`

– For `GETDATA`: Specify a source CSV file name to read data from. Specify a file name and a folder name with its relative path from the home directory of the user who logs in to the FTP server.

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- The number of characters for string data must not exceed 256.
- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the FTP server using **FP_FTP_SET_CONNECTION** or the setting dialog of the FTP client.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- The instruction can only be executed when the transfer request flag for the specified transfer setting or the specified **nLogID** number is FALSE. When the transfer request flag is TRUE, an operation error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- After the FTP client transfer settings have been completed, data is sent to files or obtained from files when **FP_FTP_TRANSFER_REQUEST** is executed.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.
- if transfer IDs are not specified in ascending order.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if an FTP server that has not been configured with the instruction **FP_FTP_SET_CONNECTION** or the setting dialog of the FTP client is specified.
- if the transfer request flag for the specified transfer setting is "TRUE: Transfer requested" when the instruction is executed.
- if the transfer request flag for a specified **nLogID** number is TRUE, e.g. if **sys_bIsLog0DataRecordingActive** is TRUE.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.
- if transfer IDs are not specified in ascending order.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if an FTP server that has not been configured with the instruction **FP_FTP_SET_CONNECTION** or the setting dialog of the FTP client is specified.
- if the transfer request flag for the specified transfer setting is "TRUE: Transfer requested" when the instruction is executed.
- if the transfer request flag for a specified **nLogID** number is TRUE, e.g. if **sys_bIsLog0DataRecordingActive** is TRUE.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

### Example

### POU header

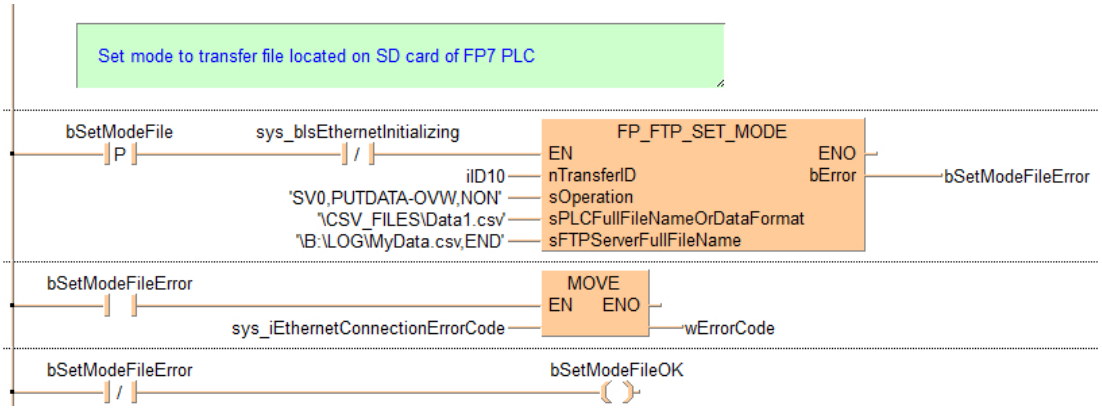
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetModeFileError | BOOL | FALSE |
| 1 | VAR | bSetModeFileOK | BOOL | FALSE |
| 2 | VAR | iID10 | INT | 10 |
| 3 | VAR | bSetModeFile | BOOL | FALSE |
| 4 | VAR | wErrorCode | WORD | 0 |

## POU body

If **bSetModeFile** changes from FALSE to TRUE and **sys_bIsEthernetInitializing** is FALSE, the instruction is carried out.

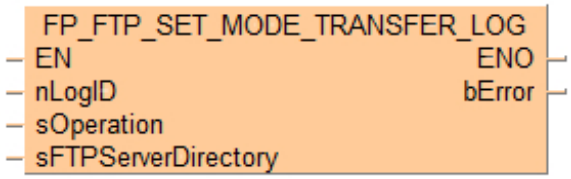
## LD body



# FP_FTP_SET_MODE_TRANSFER_LOG

Set FTP transfer mode for data recording files

This FP instruction defines the transfer settings for data recording files via FTP protocol.




## Parameters

Input

**nLogID** (WORD, INT, UINT)

LOG number (permissible range: 0–15). The compiler generates internally the string for the log file number, e.g. 'LOG=0'

**sOperation** (STRING)

Set the transfer method parameters.

- Server address (required parameter)

  Only one server can be specified at the same time. Specify an FTP server number with one-byte three characters.

  Keyword: `SV`

  Values: SV0–SV3 (Server 0–Server 3)

- Transfer method (optional parameter)

| Syntax | Transfer method |
|---|---|
| `'MODE=OVW'` | Overwrite method (default)<br>Perform file transfer with file names specified by the data recording setting. When the transfer is interrupted due to any trouble with network or servers, the file transfer may have been executed only partially. Check if the transfer has been completed successfully with an instruction such as **FP_FTP_GET_STATUS**. |
| `'MODE=REN'` | Rename method<br>Perform file transfer with temporary file names, and rename them to specified file names after the success of the transfer. To check whether the file transfer has been completed successfully, compare the actual file names with the file names specified by the data recording setting. The processing time is longer than that of the overwrite method. |

Example: Send data recording file to FTP server 3 using rename method:

`'SV3,MODE=REN'`

**sFTPServerDirectory** (STRING)

Set the destination directory (max. 256 characters). It will be created with a relative path from the home directory which has been assigned to the user after logging into the FTP server. When the destination directory does not exist, it wil be automatically created with up to eight hierarchies.

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- Separate all keyword entries by comma. e.g. 'NAME=abcd,FROM=sender@server.com'
- Do not change the order of keywords. Specify the keywords and their setting parameters in the order they are listed here.
- The number of characters for string data must not exceed 256.
- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the FTP server using **FP_FTP_SET_CONNECTION** or the setting dialog of the FTP client.
- Upper and lower case characters can be used for operands for which a character constant can be specified. "Abcd", "ABCD" and "abcd" are synonymous, however, file names are case-sensitive.
- The instruction can only be executed when the transfer request flag for the specified **nLogID** number is FALSE. When the transfer request flag is TRUE, an operation error occurs.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- The instruction stores the data recording transfer settings of **sOperation** and **sFTPServerDirectory** in the LOG file number specified by **nLogID**.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the transfer request flag for a specified **nLogID** number is TRUE, e.g. if **sys_bIsLog0DataRecordingActive** is TRUE.
- if the data recording condition of a specified **nLogID** number is not registered.
- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if an FTP server that has not been configured with the instruction **FP_FTP_SET_CONNECTION** or the setting dialog of the FTP client is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the transfer request flag for a specified **nLogID** number is TRUE, e.g. if **sys_bIsLog0DataRecordingActive** is TRUE.
- if the data recording condition of a specified **nLogID** number is not registered.
- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if an FTP server that has not been configured with the instruction **FP_FTP_SET_CONNECTION** or the setting dialog of the FTP client is specified.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

## POU header

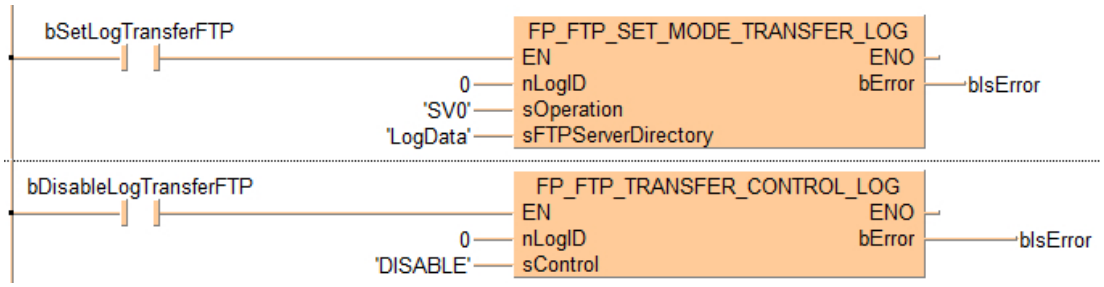
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetLogTransferFTP | BOOL | FALSE |
| 1 | VAR | bIsError | BOOL | FALSE |
| 2 | VAR | bDisableLogTransferFTP | BOOL | FALSE |

### POU body

If **bSetLogTransferFTP** and **bDisableLogTransferFTP** are set to TRUE, the instruction is carried out.

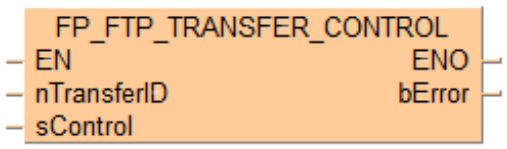
### LD body



# FP_FTP_TRANSFER_CONTROL

Control of FTP transfer to a single Ethernet unit

This FP instruction controls the FTP transfer to a single Ethernet unit specified by **nTransferID** (0–15). Valid control words are `'ENABLE'`, `'DISABLE'` and `'CANCEL'`.




## Parameters

Input

**nTransferID** (INT)

Ethernet unit ID (values: 0–15)

**sControl** (STRING)

Control string:

- `'ENABLE'`: enables file transfer to the Ethernet unit
- `'DISABLE'`: disables file transfer to the Ethernet unit
- `'CANCEL'`: cancels file transfer to the Ethernet unit

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- The number of characters for string data must not exceed 256.
- This instruction is not available in interrupt programs.
- Upper and lower case characters can be used for operands for which a character constant can be specified. "Abcd", "ABCD" and "abcd" are synonymous, however, file names are case-sensitive.
- Before you execute the instruction, you need to specify the transfer settings using **FP_FTP_SET_MODE** or the setting dialog of the FTP client.

- Before you execute the instruction, make sure that **sys_blsEthernetInitializing** is FALSE. **sys_blsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_blsEthernetInitializing** is TRUE, an error occurs.
- When this instruction has been executed successfully, the system variables **sys_blsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- It takes some time to process the transfer cancel request. Check the transfer status with **FP_FTP_GET_STATUS** and check if the transfer stops after executing the instruction.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if a transfer setting that has not been configured with **FP_FTP_SET_MODE** or in the setting dialog of the FTP client is specified.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if a transfer setting that has not been configured with **FP_FTP_SET_MODE** or in the setting dialog of the FTP client is specified.

**sys_blsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

## POU header

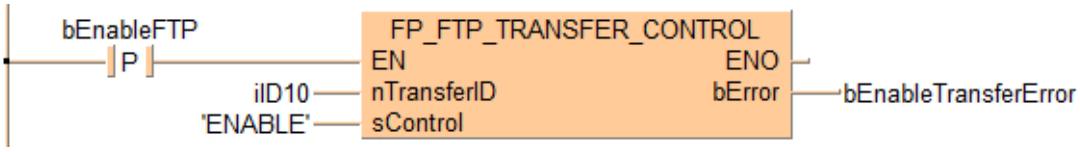
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnableFTP | BOOL | FALSE |
| 1 | VAR | bDisableTransferError | BOOL | FALSE |
| 2 | VAR | iID10 | INT | 10 |

### POU body

If **bEnableFTP** changes from FALSE to TRUE, the instruction is carried out. The control word 'ENABLE' enables the FTP transfer for the Ethernet unit 10.

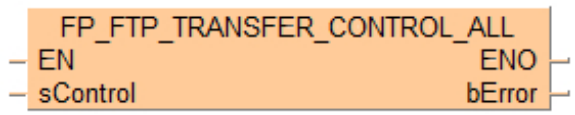
### LD body



# FP_FTP_TRANSFER_CONTROL_ALL

Control of FTP transfer to all Ethernet units

This FP instruction controls the FTP transfer to all Ethernet units. Valid control words are `'ENABLE'`, `'DISABLE'` and `'CANCEL'`.




## Parameters

Input

**sControl** (STRING)

Control string:
`'ENABLE'`: enables file transfer to all Ethernet units
`'DISABLE'`: disables file transfer to all Ethernet units
`'CANCEL'`: cancels file transfer to all Ethernet units

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- The number of characters for string data must not exceed 256.
- This instruction is not available in interrupt programs.
- Upper and lower case characters can be used for operands for which a character constant can be specified. "Abcd", "ABCD" and "abcd" are synonymous, however, file names are case-sensitive.
- Before you execute the instruction, you need to specify the transfer settings using **FP_FTP_SET_MODE** or the setting dialog of the FTP client.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.

- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- It takes some time to process the transfer cancel request. Check the transfer status with **FP_FTP_GET_STATUS** and check if the transfer stops after executing the instruction.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if a transfer setting that has not been configured with **FP_FTP_SET_MODE** or in the setting dialog of the FTP client is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if a transfer setting that has not been configured with **FP_FTP_SET_MODE** or in the setting dialog of the FTP client is specified.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

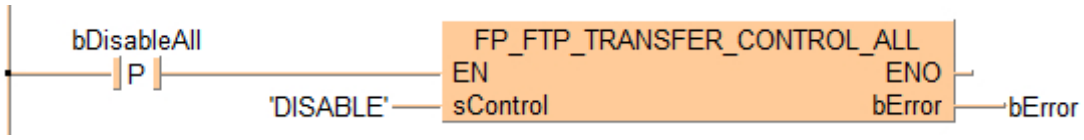
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bDisableAll | BOOL | FALSE |
| 1 | VAR | bError | BOOL | FALSE |

### POU body

If **bDisableAll** changes from FALSE to TRUE, the instruction is carried out. The control word 'DISABLE' disables the FTP transfer control to all Ethernet units.

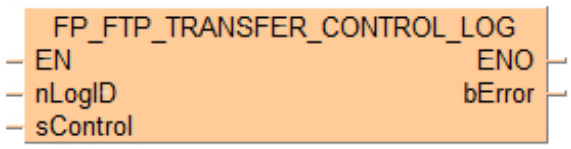
### LD body



# FP_FTP_TRANSFER_CONTROL_LOG

Control of FTP transfer of data recording files

This FP instruction controls the FTP transfer of a data recording file specified by **nLogID**. Valid control words are `'ENABLE'`, `'DISABLE'` and `'CANCEL'`.




## Parameters

Input

**nLogID** (WORD, INT, UINT)

LOG number (permissible range: 0–15). The compiler generates internally the string for the log file number, e.g. 'LOG=0'

**sControl** (STRING)

Control string:

`'ENABLE'`: enables file transfer

`'DISABLE'`: disables file transfer

`'CANCEL'`: cancels file transfer

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- The number of characters for string data must not exceed 256.
- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the transfer settings for data recording files using **FP_FTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the FTP client.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed.

When you execute the instruction while **sys_blsEthernetInitializing** is TRUE, an error occurs.

- Upper and lower case characters can be used for operands for which a character constant can be specified. "Abcd", "ABCD" and "abcd" are synonymous, however, file names are case-sensitive.
- When this instruction has been executed successfully, the system variables **sys_blsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- It takes some time to process the transfer cancel request. Check the transfer status with **FP_FTP_GET_STATUS** and check if the transfer stops after executing the instruction.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if a transfer setting for data recording files that has not been configured with the instruction **FP_FTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the FTP client is specified.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if a transfer setting for data recording files that has not been configured with the instruction **FP_FTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the FTP client is specified.

**sys_blsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

## POU header

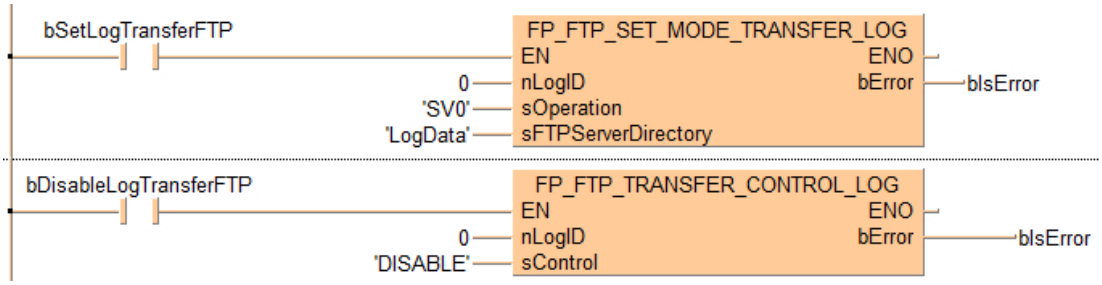
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetLogTransferFTP | BOOL | FALSE |
| 1 | VAR | bIsError | BOOL | FALSE |
| 2 | VAR | bDisableLogTransferFTP | BOOL | FALSE |

## POU body

If **bSetLogTransferFTP** and **bDisableLogTransferFTP** are set to TRUE, the instruction is carried out.

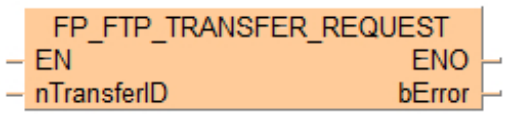
## LD body



# FP_FTP_TRANSFER_REQUEST

FTP transfer request

This FP instruction sends a request to the FTP client specified in **nTransferID** to start the transfer.




## Parameters

Input

**nTransferID** (WORD, INT, UINT)

Set the FTP client ID
Values: 0–15

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the transfer settings using **FP_FTP_SET_MODE** or the setting dialog of the FTP client.
- Before you execute the instruction, check if the system variable **sys_bIsEthernetFTPClientReady** is TRUE. If it is FALSE when executing the instruction, an error occurs.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- The instruction can only be executed when **sys_bIsEthernetCableNotConnected** is FALSE.

- The instruction can only be executed when the transfer request flag for the specified transfer setting is "FALSE: No transfer request". When the transfer request flag is "TRUE: Transfer requested", an operation error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the system variable **sys_bIsEthernetFTPClientReady** is FALSE when the instruction is executed.
- if the transfer request flag for the specified transfer setting is "TRUE: Transfer requested" when the instruction is executed.
- if a transfer setting that has not been configured with **FP_FTP_SET_MODE** or in the setting dialog of the FTP client is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the system variable **sys_bIsEthernetFTPClientReady** is FALSE when the instruction is executed.
- if the transfer request flag for the specified transfer setting is "TRUE: Transfer requested" when the instruction is executed.
- if a transfer setting that has not been configured with **FP_FTP_SET_MODE** or in the setting dialog of the FTP client is specified.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed while the Ethernet cable is disconnected. **sys_iEthernetConnectionErrorCode** is set to "10: Ethernet cable disconnected".
- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

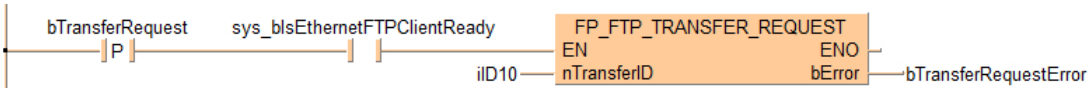
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iID10 | INT | 10 |
| 1 | VAR | bTransferRequestError | BOOL | FALSE |
| 2 | VAR | bTransferRequest | BOOL | FALSE |

### POU body

If **bTransferRequest** changes from FALSE to TRUE and the system variable **sys_bIsEthernetFTPClientReady** is TRUE, the instruction is carried out. The FTP transfer is requested for the Ethernet unit 10.

### LD body



# FP_HTTP_GET_STATUS

Get status of a single Ethernet unit using HTTP transmission

This FP instruction gets the information from the Ethernet unit specified by **nTransferID** and writes the values into the DUT **FP_CLIENT_STATUS_DUT**.


## Parameters

Input

**nTransferID** (WORD, INT, UINT)

Ethernet unit ID (values: 0–15)

Output

**dutFPClientStatus** (**FP_CLIENT_STATUS_DUT**)

Stores the values of the Ethernet status of the specified Ethernet unit

## Remarks

- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the transfer settings using **FP_HTTP_SET_MODE** or the setting dialog of the HTTP client.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the Ethernet unit ID specified is outside the permissible range (0–15)
- if a transfer setting that has not been configured with **FP_HTTP_SET_MODE** or in the setting dialog of the HTTP client is specified.
- if the instruction is executed in an interrupt program

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the Ethernet unit ID specified is outside the permissible range (0–15)
- if a transfer setting that has not been configured with **FP_HTTP_SET_MODE** or in the setting dialog of the HTTP client is specified.

- if the instruction is executed in an interrupt program

## Example

### POU header

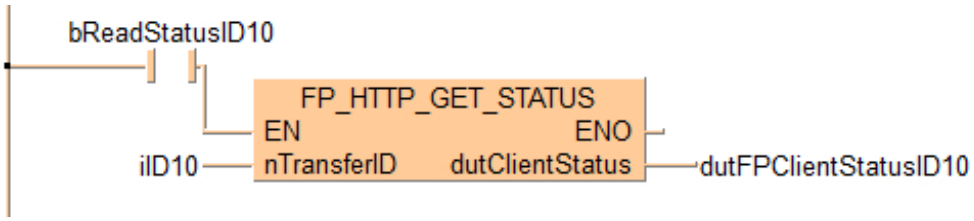
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bReadStatusID10 | BOOL | FALSE |
| 1 | VAR | iID10 | INT | 10 |
| 2 | VAR | dutFPClientStatusID10 | FP_CLIENT_STATUS_DUT | |

### POU body

If **bReadStatusID10** is set to TRUE, the instruction is carried out. It gets the HTTP transfer values from Ethernet unit 10 and writes the values into the DUT **dutFPClientStatusId10**.

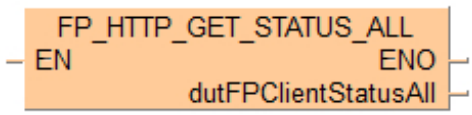
### LD body



# FP_HTTP_GET_STATUS_ALL

Get status of all Ethernet units using HTTP transmission

This FP instruction gets the information from all Ethernet units **nTransferID0**–**nTransferID15** and writes the values into the DUT **FP_CLIENT_STATUS_ALL_DUT**.




## Parameters

Output

**dutFPClientStatusAll** (**FP_CLIENT_STATUS_ALL_DUT**)

Stores the values of the Ethernet status of all Ethernet units

## Remarks

- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the transfer settings using **FP_HTTP_SET_MODE** or the setting dialog of the HTTP client.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a transfer setting that has not been configured with **FP_HTTP_SET_MODE** or in the setting dialog of the HTTP client is specified.
- if the instruction is executed in an interrupt program

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if a transfer setting that has not been configured with **FP_HTTP_SET_MODE** or in the setting dialog of the HTTP client is specified.
- if the instruction is executed in an interrupt program

## Example

### POU header

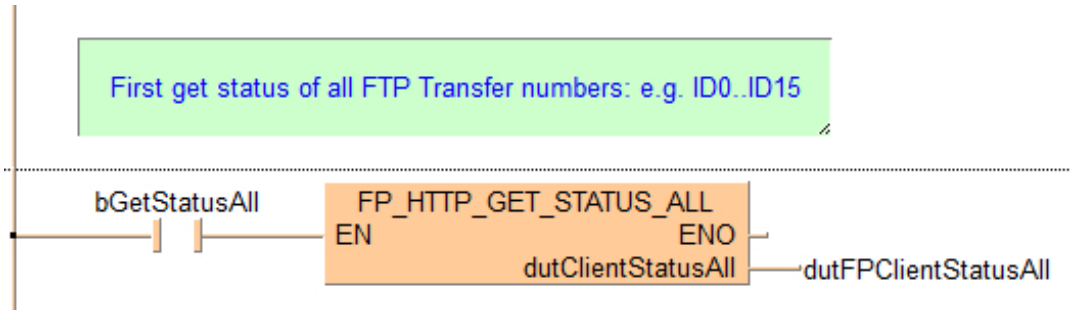
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | dutFPClientStatusAll | FP_CLIENT_STATUS_ALL_DUT | |
| 1 | VAR | bGetStatusAll | BOOL | FALSE |

### POU body

If **bGetStatusAll** is set to TRUE, the instruction is carried out. The status values of all Ethernet units are written into the DUT **dutFPClientStatusAll**.

### LD body



# FP_HTTP_SET_CONNECTION

Server settings for HTTP connection

This FP instruction defines the server settings for the connection to the HTTP client specified in the CPU according to specified parameters.


## Parameters

Input

**sServer** (STRING)

- Server address (required parameter)

  Keyword: `SV`

  Values: SV0–SV3 (Server 0–Server 3)

- IP address or host name (required parameter).
  - IP address

    For an IP address, specify the keyword `IPv4` or `IPv6` at the beginning.

    Syntax for IPv4: e.g. `'IPv4=111.122.133.144'`

    Syntax for IPv6: e.g. `'IPv6=1111:122:2:1555:0:0:1888'`

    Please note that for IPv4 addresses there are range restrictions. When an invalid IP address is specified with an instruction, there will be no operation error but the output **bError** will be set to TRUE.

  - Host name

    Keyword: `HOST`

    Syntax: e.g. `'HOST=HTTP.pidsx.com'`

- Port number (optional parameter)

  Keyword: `'PORT'`

  Syntax: `'PORT=xxxxx'`

  Values: 1–65535 (default: 80)

- SSL3/TSL1 authentication (optional parameter)

  Specify whether or not to use SSL3/TSL1 authentication.

  Keywords:
  - `SSL`: Use SSL3/TLS1
  - `NON`: SSL3/TLS1 not used (default)

Examples:

1. Connect to HTTP server number 0 with the IP address 192.255.2.10, port number 80, using SSL3/TLS1 authentication: `'SV0,IPv4=192.255.2.10,PORT=80,SSL'`
2. Connect to HTTP server number 1 with the IP address 1111:1222::1555:0:0:1888, port number 8080, using SSL3/TLS1 authentication: `'SV1,IPv6=1111:1222::1555:0:0:1888,PORT=8080,SSL'`
3. Connect to HTTP server number 2 with the host name HTTP.pidsx.com, port number 80, do not use authentication: `'SV2,HOST=HTTP.pidsx.com,PORT=80,NON'`

**sLogin** (STRING)

Set the login data

- User name (max. 32 characters)

  Keyword: `USER`

  Syntax: `'USER=xxx'` (default: root). Use `'USER='` to delete the user name.
- Password (max. 32 characters, upper and lower case characters are allowed)

  Keyword: `PASS`

  Syntax: `'PASS=passwd'` (default: root). Use `'PASS='` to delete the password.

For this parameter, there are two additional keywords available:

- `INITIAL`: Resets user name and password to the default settings "root" and "root".
- `KEEP`: Keeps the current login settings.

Examples:

1. Set user name to "Admin" and password to "Panasonic": `'USER=Admin,PASS=Panasonic'`
2. Set user name to "Supervisor" and delete the password: `'USER=Supervisor,PASS='`
3. Set user name to "Support", do not change the password: `'USER=Support'`
4. Delete user name and password: `'USER=,PASS='`
5. Reset user name and password to the default values: `'INITIAL'`
6. Keep the current user name and password: `'KEEP'`

**sCommunication** (STRING)

Set optional communication parameters as required.

- Connection timeout

  Keyword: `TOUT`

  Syntax:`'TOUT=xxx'` (default: 60 seconds)

  Values: 30–300 seconds
- Number of retries

  Keyword: `RTRY`

  Syntax: `'RTRY=x'` (default: 3 times)

  Values: 0–3
- Retry interval

  Keyword: `RTTM`

Syntax: `'RTTM=xxxxx'` (default: 600 seconds)

Values: 10–86400 seconds

The value can be specified by 10 seconds. It is rounded down to the 10. Example: When you specify 38 seconds, 30 seconds are set.

For this parameter, there are two additional keywords available:

- `INITIAL`: Resets connection timeout, number of retries, and the retry interval to the default settings.
- `KEEP`: Keeps the current communication settings.

Examples:

1. Set connection timeout: 30 seconds, number of retries: 2, retry interval: 500 seconds: `'TOUT=30,RTRY=2,RTTM=500'`
2. Set connection timeout: 270 seconds, no retries, retry interval: 4900 seconds: `'TOUT=270,RTRY=0,RTTM=4900'`
3. Set connection timeout: 30 seconds, number of retries: 25, retry interval: do not change: `'TOUT=30,RTRY=25'`
4. Set connection timeout: do not change, number of retries: 25, retry interval: 3000 seconds: `',RTRY=25,RTTM=3000'`
5. Reset to default settings (connection timeout: 60 seconds, number of retries: 3, retry interval: 600 seconds): `'INITIAL'`
6. Keep all the current settings: `'KEEP'`

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- Separate all keyword entries by comma. e.g. 'NAME=abcd,FROM=sender@server.com'
- The number of characters for string data must not exceed 256.
- This instruction is not available in interrupt programs.
- Do not change the order of keywords. Specify the keywords and their setting parameters in the order they are listed here.
- A part of the parameter syntax can be omitted. The settings are not changed when parameters are omitted partially.
- When omitting the part **before** a specified keyword, omit only the keyword, but not the comma "," separating the keywords: `',RTRY=25,RTTM=3000'`.

- When omitting the part **after** a specified keyword, omit both the comma "," and the keyword: `'TOUT=30,RTRY=25'`.
- Do not specify the same keyword more than once. If the same keyword is specified more than once, an error occurs.
- Upper and lower case characters can be used for operands for which a character constant can be specified. "Abcd", "ABCD" and "abcd" are synonymous, however, file names are case-sensitive.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- The instruction can only be executed when the transfer request flag for the specified transfer setting is "FALSE: No transfer request". When the transfer request flag is "TRUE: Transfer requested", an operation error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.
- if the same keyword is specified more than once
- if the transfer request flag for the specified transfer setting is "TRUE: Transfer requested" when the instruction is executed.
- if server numbers are not specified in ascending order.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.
- if the same keyword is specified more than once
- if the transfer request flag for the specified transfer setting is "TRUE: Transfer requested" when the instruction is executed.
- if server numbers are not specified in ascending order.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed with an incorrect IP address, **sys_iEthernetConnectionErrorCode** is set to "1: Incorrect IP address is specified"

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

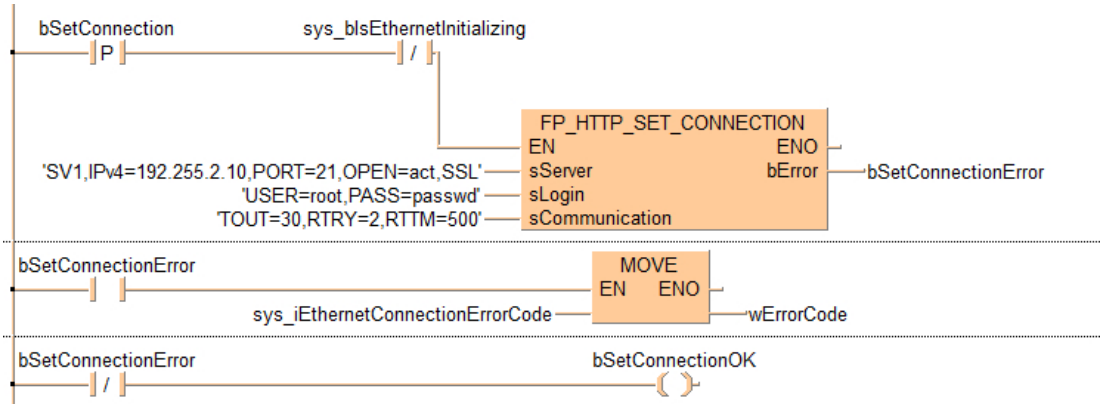
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetConnectionError | BOOL | FALSE |
| 1 | VAR | bSetConnection | BOOL | FALSE |
| 2 | VAR | wErrorCode | WORD | 0 |
| 3 | VAR | bSetConnectionOK | BOOL | FALSE |

### POU body

If **bSetConnection** changes from FALSE to TRUE and **sys_bIsEthernetInitializing** is FALSE, the instruction is carried out.

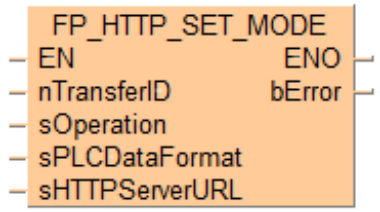
### LD body



# FP_HTTP_SET_MODE

Set HTTP transfer mode

This FP instruction stores the HTTP client transfer settings of **sOperation**, **sPLCDataFormat** and **sHTTPServerURL** in the transfer setting area specified by **nTransferID**.




## Parameters

Input

**nTransferID** (WORD, INT, UINT)

Set the transfer setting ID.
Values: 0–15 (values must be entered in ascending order)

**sOperation** (STRING)

Set the transfer method parameters.

- Server address

  Keyword: `SV`

  Values: SV0–SV3 (Server 0–Server 3)

- Specify the memory area and the type of transfer (sending or getting data)
  - Use the keyword `UPLOAD` for sending data.
  - Use the keyword `DOWNLOAD` for getting data.
  - Use the keyword `UPDOWN` for sending and getting data.

  Specify the command to be used for transfer.
  - Use the keyword `POST` for sending data (only available for the transfer types UPLOAD and UPDOWN).
  - Use the keyword `GET` for getting data.

  Example: Upload data from memory area to HTTP server 3: `'SV3,UPDOWN,POST'`

**sPLCDataFormat** (STRING)

Specify the data to be transferred. Alternatively, use the instruction **FP_HTTP_GET_DATA_FORMAT.**

Example: Memory area: DT100000, number of bytes: 250 bytes, maximum number of acquisitions: 250 bytes: `'DT100000,250,250'`

**sHTTPServerURL** (STRING)

Specify the server URL.
Example: `'\data.csv'`

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- The number of characters for string data must not exceed 256.
- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the HTTP server using **FP_HTTP_SET_CONNECTION** or the setting dialog of the HTTP client.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- The instruction can only be executed when the transfer request flag for the specified transfer setting is "FALSE: No transfer request". When the transfer request flag is "TRUE: Transfer requested", an operation error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- After the HTTP client transfer settings have been configured, data is actually sent or acquired when the instruction **FP_HTTP_TRANSFER_REQUEST** is executed.
- The number of bytes that can be simultaneously transferred is 1MB for all 16 transfer IDs.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.
- if transfer IDs are not specified in ascending order.
- if the instruction is executed in an interrupt program

- if the number of characters for string data exceeds 256.
- if an HTTP server that has not been configured with the instruction **FP_HTTP_SET_CONNECTION** or the setting dialog of the HTTP client is specified.
- if the transfer request flag for the specified transfer setting is "TRUE: Transfer requested" when the instruction is executed.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.
- if transfer IDs are not specified in ascending order.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if an HTTP server that has not been configured with the instruction **FP_HTTP_SET_CONNECTION** or the setting dialog of the HTTP client is specified.
- if the transfer request flag for the specified transfer setting is "TRUE: Transfer requested" when the instruction is executed.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

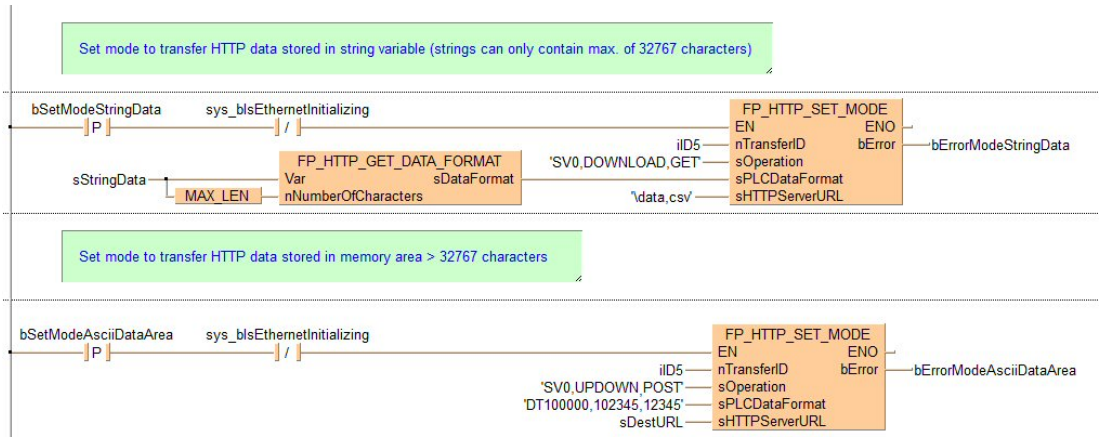
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | sDestURL | STRING[32] | '\data.csv' |
| 2 | VAR | sStringData | STRING[32767] | '' |
| 3 | VAR | iID5 | INT | 5 |
| 4 | VAR | bSetModeAsciiDataArea | BOOL | FALSE |
| 5 | VAR | bSetModeStringData | BOOL | FALSE |
| 6 | VAR | bErrorModeStringData | BOOL | FALSE |
| 7 | VAR | bErrorModeAsciiDataArea | BOOL | FALSE |

## POU body

The first example uses a **source** string containing a maximum of 32767 characters, the second example uses a DT address to store the **source** string with more than 32767 characters.

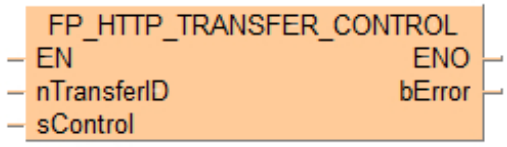
## LD body



# FP_HTTP_TRANSFER_CONTROL

Control of HTTP transfer to a single Ethernet unit

This FP instruction controls the HTTP transfer to a single Ethernet unit specified by **nTransferID** (0–15). Valid control words are `'ENABLE'`, `'DISABLE'` and `'CANCEL'`.




## Parameters

Input

**nTransferID** (WORD, INT, UINT)

Ethernet unit ID (values: 0–15)

**sControl** (STRING)

Control string:

- `'ENABLE'`: enables HTTP transfer to the Ethernet unit
- `'DISABLE'`: disables HTTP transfer to the Ethernet unit
- `'CANCEL'`: cancels HTTP transfer to the Ethernet unit

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- The number of characters for string data must not exceed 256.
- This instruction is not available in interrupt programs.
- Upper and lower case characters can be used for operands for which a character constant can be specified. "Abcd", "ABCD" and "abcd" are synonymous, however, file names are case-sensitive.
- Before you execute the instruction, you need to specify the transfer settings using **FP_HTTP_SET_MODE** or the setting dialog of the HTTP client.

- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- It takes some time to process the transfer cancel request. Check the transfer status with **FP_HTTP_GET_STATUS** and check if the transfer stops after executing the instruction.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if a transfer setting that has not been configured with **FP_HTTP_SET_MODE** or in the setting dialog of the HTTP client is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if a transfer setting that has not been configured with **FP_HTTP_SET_MODE** or in the setting dialog of the HTTP client is specified.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

## POU header

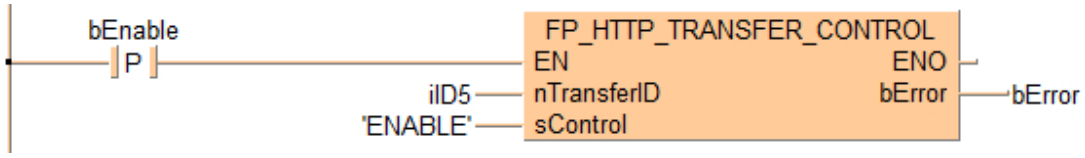
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | bError | BOOL | FALSE |
| 2 | VAR | iID5 | INT | 5 |

## POU body

If **bEnable** changes from FALSE to TRUE, the instruction is carried out. The control word 'ENABLE' enables the HTTP transfer for the Ethernet unit 5.

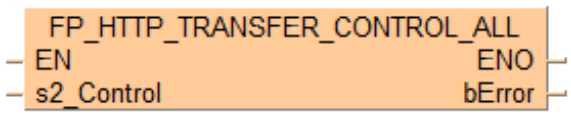
## LD body



# FP_HTTP_TRANSFER_CONTROL_ALL

Control of HTTP transfer to all Ethernet units

This FP instruction controls the HTTP transfer to all Ethernet units. Valid control words are `'ENABLE'`, `'DISABLE'` and `'CANCEL'`.




## Parameters

Input

**s2_Control** (STRING)

Control string:

- `'ENABLE'`: enables HTTP transfer to all Ethernet units
- `'DISABLE'`: disables HTTP transfer to all Ethernet units
- `'CANCEL'`: cancels HTTP transfer to all Ethernet units

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- The number of characters for string data must not exceed 256.
- This instruction is not available in interrupt programs.
- Upper and lower case characters can be used for operands for which a character constant can be specified. "Abcd", "ABCD" and "abcd" are synonymous, however, file names are case-sensitive.
- Before you execute the instruction, you need to specify the transfer settings using **FP_HTTP_SET_MODE** or the setting dialog of the HTTP client.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.

- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- It takes some time to process the transfer cancel request. Check the transfer status with **FP_HTTP_GET_STATUS** and check if the transfer stops after executing the instruction.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if a transfer setting that has not been configured with **FP_HTTP_SET_MODE** or in the setting dialog of the HTTP client is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if a transfer setting that has not been configured with **FP_HTTP_SET_MODE** or in the setting dialog of the HTTP client is specified.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

## POU header

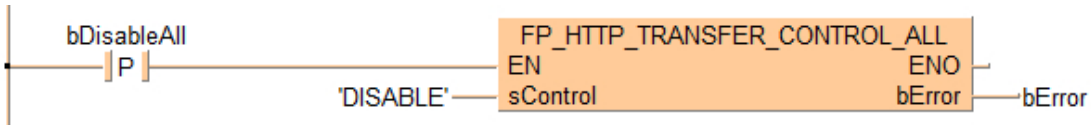
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bDisableAll | BOOL | FALSE |
| 1 | VAR | bError | BOOL | FALSE |

### POU body

If **bDisableAll** changes from FALSE to TRUE, the instruction is carried out. The control word 'DISABLE' disables the HTTP transfer to all Ethernet units.

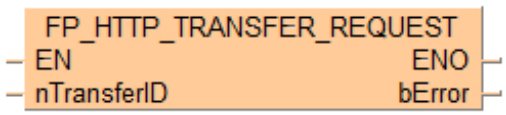
### LD body



# FP_HTTP_TRANSFER_REQUEST

HTTP transfer request

This FP instruction sends a request to the HTTP client specified in **nTransferID** to start the transfer.




## Parameters

Input

**nTransferID** (WORD, INT, UINT)

Set the HTTP client ID
Values: 0–15

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

  When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the transfer settings using **FP_HTTP_SET_MODE** or the setting dialog of the HTTP client.
- Before you execute the instruction, check if the system variable **sys_bIsEthernetHTTPClientReady** is TRUE. If it is FALSE when executing the instruction, an error occurs.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- The instruction can only be executed when **sys_bIsEthernetCableNotConnected** is FALSE.

- The instruction can only be executed when the transfer request flag for the specified transfer setting is "FALSE: No transfer request". When the transfer request flag is "TRUE: Transfer requested", an operation error occurs.
- When this instruction has been executed successfully, the system variables **sys_blsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When you try to execute the instruction under one of the following conditions, a transfer error occurs and the error code is stored.

| Status | Code |
|---|---|
| Destination server is not set. | 1 |
| Transfer setting is not set. | 2 |
| Registering a process request failed. | 4 |
| Transfer has been disabled | 5 |
| Data decompression failed. (When accessing data with PUT) | 8 |
| Data decompression failed. (When accessing data with GET) | 9 |

- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program.
- if the system variable **sys_blsEthernetHTTPClientReady** is FALSE when the instruction is executed.
- if the transfer request flag for the specified transfer setting is "TRUE: Transfer requested" when the instruction is executed.
- if a transfer setting that has not been configured with **FP_HTTP_SET_MODE** or in the setting dialog of the HTTP client is specified.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program.
- if the system variable **sys_blsEthernetHTTPClientReady** is FALSE when the instruction is executed.
- if the transfer request flag for the specified transfer setting is "TRUE: Transfer requested" when the instruction is executed.
- if a transfer setting that has not been configured with **FP_HTTP_SET_MODE** or in the setting dialog of the HTTP client is specified.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed while the Ethernet cable is disconnected. **sys_iEthernetConnectionErrorCode** is set to "10: Ethernet cable disconnected".
- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

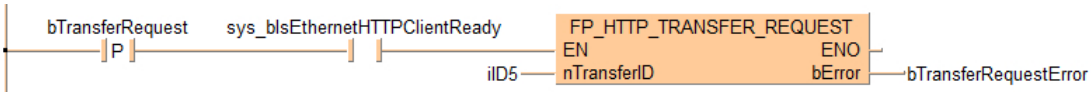
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iID10 | INT | 10 |
| 1 | VAR | bTransferRequestError | BOOL | FALSE |
| 2 | VAR | bTransferRequest | BOOL | FALSE |

### POU body

If **bTransferRequest** changes from FALSE to TRUE and the system variable **sys_bIsEthernetHTTPClientReady** is TRUE, the instruction is carried out. The HTTP transfer is requested for the Ethernet unit 5.

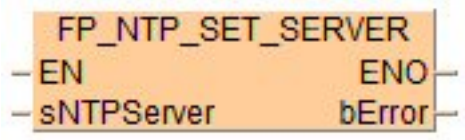
### LD body



# FP_NTP_SET_SERVER

Set NTP server connection

This FP instruction sets a destination NTP server to which a time synchronization request can be sent.




## Parameters

Input

**sNTPServer** (STRING)

IP address or host name (required parameter).

- IP address

  For an IP address, specify the keyword `IPv4` or `IPv6` at the beginning.

  Syntax for IPv4: e.g. `'IPv4=111.122.133.144'`

  Syntax for IPv6: e.g. `'IPv6=1111:122:2:1555:0:0:1888'`

- Host name

  Keyword: `HOST`

  Syntax: e.g.`'HOST=ntp.pidsx.com'`

Output

**bError** (BOOL)

Set to TRUE when the operation ends abnormally without execution, e.g. when **sys_bIsEthernetInitializing** is TRUE
When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- This instruction is not available in interrupt programs.
- The NTP server address is set in the CPU unit (built-in ET-LAN) according to **sNTPServer**.
- The data that is already set in the system registers (“Ethernet” > “Time synchronization”) is invalid and the NTP time synchronisation request is executed at the time that is specified by this instruction.
- The PLC returns to the settings from the system registers in the following cases:
  - The power is turned off.

  - You switch from “PROG mode” to “RUN mode”.
  - You switch to a different project.
- The setting data will not be lost even if the instruction **FP_IPV4_SET_ADDRESS** is executed.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When the instruction has not been executed successfully, check the execution result code in output **nResult** of the **FP_NTP_SYNCHRONIZE** instruction.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the system is synchronizing the time with the NTP server.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the system is synchronizing the time with the NTP server.

**sys_bIsCarry** (turns to TRUE for one scan)

if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | sNTPServer | STRING[32] | 'IPv4=192.168.2.1' |
| 2 | VAR | bIsError | BOOL | FALSE |
| 3 | VAR | bEnable | BOOL | FALSE |

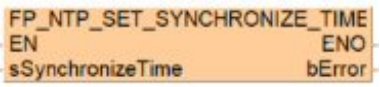
## LD body

When the system variable **sys_bIsEthernetInitializing** is FALSE and the variable **bEnable** is set to TRUE, the function is carried out.

# FP_NTP_SET_SYNCHRONIZE_TIME

Set synchronization time for NTP server

This FP instruction sets the time for executing an automatic time synchronization with the NTP server.




## Parameters

Input

**sSynchronizeTime** (STRING)

Specifies the time for automatic time synchronization with the NTP server.
Set the time with one or more of the following keyword settings:

- Daily synchronization (optional parameter)

  Keyword: `DAY`

  Syntax: `'DAY=xxx'`

  Values: DISABLE, HHMM

  HH=hours (0–23), MM=minutes (0–59)

  Examples:

  1. No daily synchronization: `'DAY=DISABLE'`
  2. Daily at 13:30: `'DAY=1330'`

- Weekly synchronization (optional parameter)

  Keyword: `WEEK`

  Syntax: `'WEEK=xxxxx'`

  Values: DISABLE, WHHMM

  W=day of week (0=Sunday, 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday), HH=hours (0–23), MM=minutes (0–59)

  Examples:

  1. No weekly synchronization: `'WEEK=DISABLE'`
  2. Every Sunday at 23:59: `'WEEK=02359'`

- Monthly synchronization (optional parameter)

  Keyword: `MONTH`

  Syntax: `'MONTH,xxxxxx'`

  Values: DISABLE, DDHHMM

  DD=day (0–28), HH=hours (0–23), MM=minutes (0–59)

  Examples:

  1. No monthly synchronization: `'MONTH=DISABLE'`

2. On every 15th day of the month at 15:30: `'MONTH=150330'`

Output

**bError** (BOOL)

Set to TRUE when the operation ends abnormally without execution, e.g. when **sys_blsEthernetInitializing** is TRUE
When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- This instruction is not available in interrupt programs.
- Do not change the order of keywords. Specify the keywords and their setting parameters in the order they are listed here.
- A part of the parameter syntax can be omitted. The settings are not changed when parameters are omitted partially.
- When omitting the part **before** a specified keyword, omit only the keyword, but not the comma "," separating the keywords: `',,MONTH=150300'`.
- When omitting the part **after** a specified keyword, omit both the comma "," and the keyword: `'DAY=1130,WEEK=DISABLE'`.
- Do not specify the same keyword more than once. If the same keyword is specified more than once, an error occurs.
- The timing of the time synchronization request for the NTP server is set according to **sSynchronizeTime**.
- The data that is already set in the system registers (“Ethernet” > “Time synchronization”) is invalid and the NTP time synchronisation request is executed at the time that is specified by this instruction.
- The PLC returns to the settings from the system registers in the following cases:
  - The power is turned off.
  - You switch from “PROG mode” to “RUN mode”.
  - You switch to a different project.
- The setting data will not be lost even if the instruction **FP_IPV4_SET_ADDRESS** is executed.
- Before you execute the instruction, make sure that **sys_blsEthernetInitializing** is FALSE. **sys_blsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_blsEthernetInitializing** is TRUE, an error occurs.
- When this instruction has been executed successfully, the system variables **sys_blsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the system is synchronizing the time with the NTP server.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the system is synchronizing the time with the NTP server.

**sys_bIsCarry** (turns to TRUE for one scan)

if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

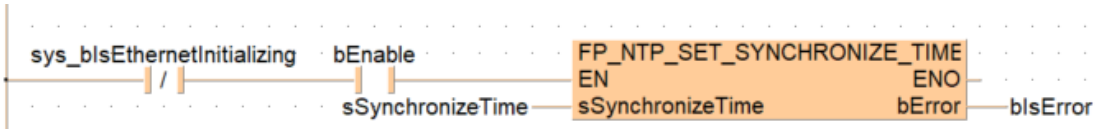
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | sSynchronizeTime | STRING[32] | 'DAY=1130,WEEK=62345,MONTH=280200' |
| 2 | VAR | bIsError | BOOL | FALSE |
| 3 | VAR | bEnable | BOOL | FALSE |

### LD body

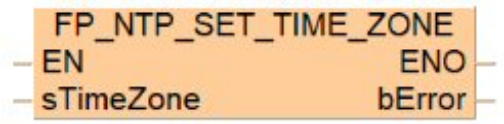
When the system variable **sys_bIsEthernetInitializing** is FALSE and the variable **bEnable** is set to TRUE, the function is carried out.



# FP_NTP_SET_TIME_ZONE

Set local time zone of NTP server

This FP instruction is only necessary if you want to temporarily set a local time zone in the PLC to replace the time zone set by the system register under “Ethernet” > “Time synchronization”. This local time zone is specified by **sTimeZone** and is used in combination with the time supplied by **FP_NTP_SYNCHRONIZE**.




## Parameters

Input

**sTimeZone** (STRING)

Specifies the time zone based on GMT (Greenwich Mean Time).
Syntax: `+HHMM/-HHMM`
Values: format: sign (+/-) and HHMM (HH=hours (00–23), MM=minutes (00–59))
Examples:

1. For France, Italy, Spain, Germany: `'+0100'`
2. For Japan: `'+0900'`
3. For Hawaii: `'-1000'`

Output

**bError** (BOOL)

Set to TRUE when the operation ends abnormally without execution, e.g. when **sys_bIsEthernetInitializing** is TRUE
When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- This instruction is not available in interrupt programs.
- The PLC returns to the settings from the system registers in the following cases:
  - The power is turned off.
  - You switch from “PROG mode” to “RUN mode”.
  - You switch to a different project.
- The setting data will not be lost even if the instruction **FP_IPV4_SET_ADDRESS** is executed.

- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the system is synchronizing the time with the NTP server.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the system is synchronizing the time with the NTP server.

**sys_bIsCarry** (turns to TRUE for one scan)

if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | sTimeZone | STRING[32] | '+0100' |
| 2 | VAR | bIsError | BOOL | FALSE |
| 3 | VAR | bEnable | BOOL | FALSE |

### LD body

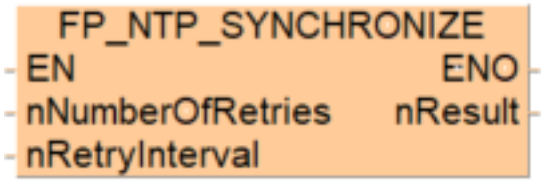
When the system variable **sys_bIsEthernetInitializing** is FALSE and the variable **bEnable** changes from FALSE to TRUE, the function is carried out.


sys_bIsEthernetInitializing
/
bEnable
P
sTimeZone
FP_NTP_SET_TIME_ZONE
EN
ENO
sTimeZone
bError
bIsError

# FP_NTP_SYNCHRONIZE

Request time synchronization from NTP server

This FP instruction sends a request to the NTP server to synchronize the time.




## Parameters

Input

**nNumberOfRetries** (WORD, INT, UINT)

Number of retries of requesting the time synchronization
Values: 0–20

**nRetryInterval** (WORD, INT, UINT)

Time synchronization interval
Values: 16–600

Output

**nResult** (WORD, INT, UINT)

Execution result code

| | | |
|---|---|---|
| 16#FFFF | In progress | |
| 16#0 | Normal completion | |
| 16#10 | Double startup error | The instruction requesting the time synchronization is being executed.<br>Note<br>The double startup error does not occur when the instruction is executed with the number of retries being set to 0 to cancel the time synchronization request instruction. |
| 16#11 | SNTP server address setting error | ET-LAN setting, SNTP server address setting = "0.0.0.0" |
| 16#12 | Disconnection error | Ethernet is disconnected. |

| | | |
|---|---|---|
| 16#13 | Ethernet initialization active error | No IP address has been assigned to the Ethernet unit sending the request. (**sys_blsEthernetIPAddressAssigned** is FALSE) |
| 16#14 | Number of retries setting error | The specified number of retries is out of range. |
| 16#15 | Retry interval setting error | The specified retry interval is out of range. |
| 16#20 | Response timeout error | The response time of processing the time synchronization request exceeds the predefined time.<br>Note<br>This error also occurs when an NTP IP address is not resolved. |
| 16#30 | Ethernet task response timeout | This error occurs when no response is returned from the Ethernet task. |

## Remarks

- This instruction is not available in interrupt programs.
- If a time synchronization timeout is foreseeable, set **nNumberOfRetries** to a higher value.
- For cancelling further attempts to synchronize the time, set **nNumberOfRetries** to 0. In this case, the execution result code is not stored in **nResult**.
- The timeout period for one time synchronization attempt is fixed at 3 seconds.
- When multiple time synchronization attempts are specified, a new request starts after the timeout period elapses (3 seconds) plus the processing interval (specified by **nRetryInterval**).
- The total timeout period (seconds) for time synchronization is obtained with the following formula: **nRetryInterval** x 3 + (**nRetryInterval** x (**nNumberOfRetries**-1)). (Here, **nNumberOfRetries** is larger than 0.)

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the instruction is executed in an interrupt program

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the instruction is executed in an interrupt program

## Example

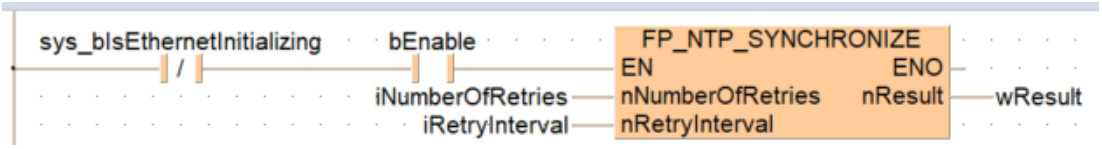
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bRequestSynchronization | BOOL | FALSE |
| 2 | VAR | iNumberOfRetries | INT | 3 |
| 3 | VAR | iRetryInterval | INT | 20 |
| 4 | VAR | wResult | WORD | 0 |
| 5 | VAR | bEnable | BOOL | FALSE |

### LD body

When the system variable **sys_bIsEthernetInitializing** is FALSE and the variable **bEnable** is set to TRUE, the function is carried out.



# FP_NTP_SET_RETRY_SETTINGS

Set retry settings for NTP server

This FP instruction sets the retry settings for NTP server time synchronization requests.


## Parameters

Input

**nTimeout** (WORD, INT, UINT)

Timeout in seconds
Values: 30–300 seconds

**nNumberOfRetries** (WORD, INT, UINT)

Number of retries
Values: 0–3

**nRetryInterval** (DWORD, DINT, UDINT, DATE, TOD, DT)

Length of an interval for a retry in seconds
Values: 10–86400 seconds

Output

**bError** (BOOL)

Set to TRUE when the operation ends abnormally without execution, e.g. when **sys_bIsEthernetInitializing** is TRUE
When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- This instruction is not available in interrupt programs.
- The retry settings of the time synchronization request for the NTP server is set according to **nTimeout**, **nNumberOfRetries** and **nRetryInterval**.
- The data that is already set in the system registers (“Ethernet” > “Time synchronization”) is invalid and the NTP time synchronisation request is executed at the time that is specified by this instruction.
- The PLC returns to the settings from the system registers in the following cases:

  - The power is turned off.
  - You switch from “PROG mode” to “RUN mode”.
  - You switch to a different project.
- The setting data will not be lost even if the instruction **FP_IPV4_SET_ADDRESS** is executed.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the system is synchronizing the time with the NTP server.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the system is synchronizing the time with the NTP server.

**sys_bIsCarry** (turns to TRUE for one scan)

if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

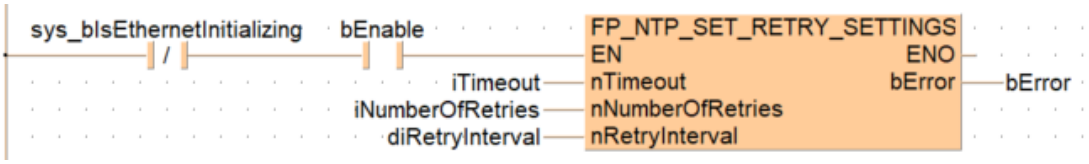
## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | bError | BOOL | FALSE |
| 3 | VAR | iTimeout | INT | 60 |
| 4 | VAR | iNumberOfRetries | INT | 3 |
| 5 | VAR | diRetryInterval | DINT | 600 |

## LD body

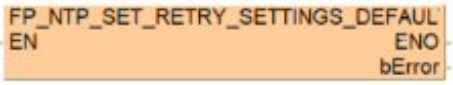
When the system variable **sys_bIsEthernetInitializing** is FALSE and the variable **bEnable** is set to TRUE, the function is carried out.



# FP_NTP_SET_RETRY_SETTINGS_DEFAULT

Set retry settings to default values

This FP instruction sets the retry settings for NTP server time synchronization requests to the default values.




## Parameters

Output

**bError** (BOOL)

Set to TRUE when the operation ends abnormally without execution, e.g. when **sys_bIsEthernetInitializing** is TRUE
When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- This instruction is not available in interrupt programs.
- The retry settings of the time synchronization request for the NTP server will be set to the following default values:
  Timeout period: 60 seconds
  Number of retries: 3
  Retry interval: 600 seconds
- The data that is already set in the system registers (“Ethernet” > “Time synchronization”) is invalid and the NTP time synchronisation request is executed at the time that is specified by this instruction.
- The PLC returns to the settings from the system registers in the following cases:
  – The power is turned off.
  – You switch from “PROG mode” to “RUN mode”.
  – You switch to a different project.
- The setting data will not be lost even if the instruction **FP_IPV4_SET_ADDRESS** is executed.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.

- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the system is synchronizing the time with the NTP server.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the system is synchronizing the time with the NTP server.

**sys_bIsCarry** (turns to TRUE for one scan)

if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

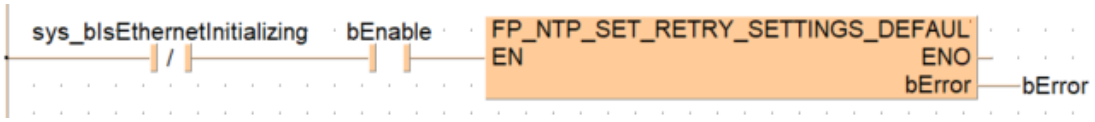
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bError | BOOL | FALSE |
| 2 | VAR | bEnable | BOOL | FALSE |

### LD body

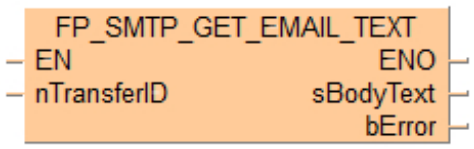
When the system variable **sys_bIsEthernetInitializing** is FALSE and the variable **bEnable** is set to TRUE, the function is carried out.



# FP_SMTP_GET_EMAIL_TEXT

Get e-mail text

This FP instruction reads the predefined e-mail texts of the number specified by **nTransferID** and writes the text into the output variable **sBodyText**.




## Parameters

Input

**nTransferID** (WORD, INT, UINT)

Set the transfer setting ID.
Values: 0–15

Output

**sBodyText** (STRING)

Stores the predefined e-mail text

**bError** (BOOL)

Set to TRUE when the operation ends abnormally.
When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the e-mail transmission settings using **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client.
- This instruction reads the e-mail text that has been specified in the setting dialog of the SMTP client. If no e-mail text has been specified, it cannot be read and zero is stored in **sBodyText**.
- Before you execute the instruction, you need to specify the group and event e-mail settings using **FP_SMTP_SET_GROUP** or the setting dialog of the SMTP client.
- Before you execute the instruction, check **FP_CLIENT_STATUS_DUT** whether there is an active request to send an e-mail. If **bIsTransferRequested** is TRUE when you execute the instruction, an operation error occurs.

- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program.
- if the e-mail transmission request flag for the specified transfer setting is "TRUE: Transfer requested".
- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.
- if a destination group number that has not been defined with the instruction **FP_SMTP_SET_GROUP** or the setting dialog of the SMTP client is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program.
- if the e-mail transmission request flag for the specified transfer setting is "TRUE: Transfer requested".
- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.
- if a destination group number that has not been defined with the instruction **FP_SMTP_SET_GROUP** or the setting dialog of the SMTP client is specified.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

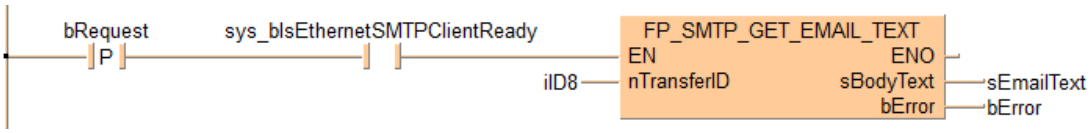
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bRequest | BOOL | FALSE |
| 1 | VAR | bError | BOOL | FALSE |
| 2 | VAR | iID8 | INT | 8 |
| 3 | VAR | sEmailText | STRING[32] | 'Hello World!' |

### POU body

If **bRequest** changes from FALSE to TRUE and **sys_bIsEthernetSMTPClientReady** is set to TRUE, the instruction is carried out.

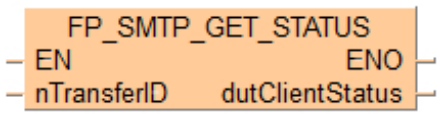
### LD body



# FP_SMTP_GET_STATUS

Get status of a single Ethernet unit using SMTP transmission

This FP instruction gets the information from the Ethernet unit specified by **nTransferID** and writes the values into the DUT **FP_CLIENT_STATUS_DUT**.




## Parameters

Input

**nTransferID** (WORD, INT, UINT)

Ethernet unit ID (values: 0–15)

Output

**dutClientStatus** (**FP_CLIENT_STATUS_DUT**)

Stores the values of the Ethernet status of the specified Ethernet unit

## Remarks

- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the e-mail transmission settings using **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client.
- Before you execute the instruction, you need to specify the e-mail transfer settings for data recording files using **FP_SMTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the SMTP client.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the Ethernet unit ID specified is outside the permissible range (0–15)
- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.
- if an e-mail transmission setting for data recording files that has not been configured with the instruction **FP_SMTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the SMTP client is specified.
- if the instruction is executed in an interrupt program

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the Ethernet unit ID specified is outside the permissible range (0–15)
- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.
- if an e-mail transmission setting for data recording files that has not been configured with the instruction **FP_SMTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the SMTP client is specified.
- if the instruction is executed in an interrupt program

## Example

### POU header

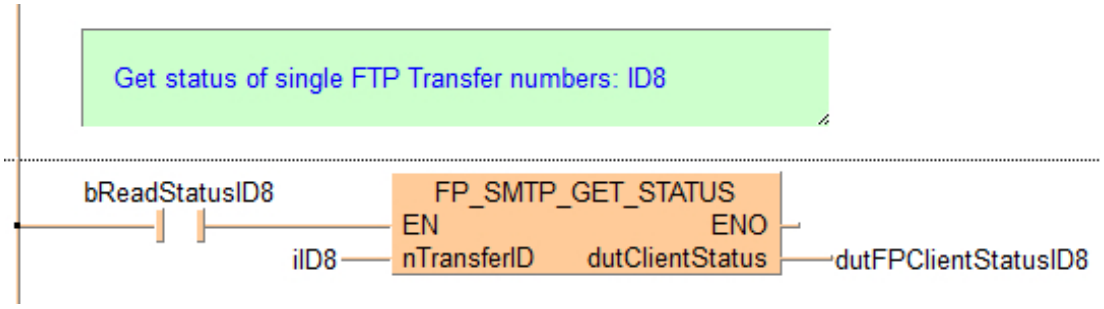
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | dutFPClientStatusId5 | FP_CLIENT_STATUS_DUT | |
| 1 | VAR | iID8 | INT | 8 |
| 2 | VAR | bReadStatusID8 | BOOL | FALSE |

### POU body

If **bReadStatusID8** is set to TRUE, the instruction is carried out.

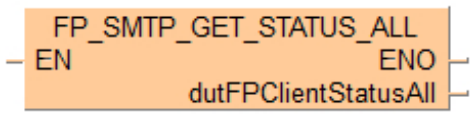
### LD body



# FP_SMTP_GET_STATUS_ALL

Get status of all Ethernet units using SMTP transmission

This FP instruction gets the information from all Ethernet units **nTransferID0**–**nTransferID15** and writes the values into the DUT **FP_CLIENT_STATUS_ALL_DUT**.




## Parameters

Output

**dutFPClientStatusAll (FP_CLIENT_STATUS_ALL_DUT)**

Stores the values of the Ethernet status of all Ethernet units

## Remarks

- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the e-mail transmission settings using **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client.
- Before you execute the instruction, you need to specify the e-mail transfer settings for data recording files using **FP_SMTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the SMTP client.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.
- if an e-mail transmission setting for data recording files that has not been configured with the instruction **FP_SMTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the SMTP client is specified.
- if the instruction is executed in an interrupt program

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.

- if an e-mail transmission setting for data recording files that has not been configured with the instruction **FP_SMTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the SMTP client is specified.
- if the instruction is executed in an interrupt program

## Example

### POU header

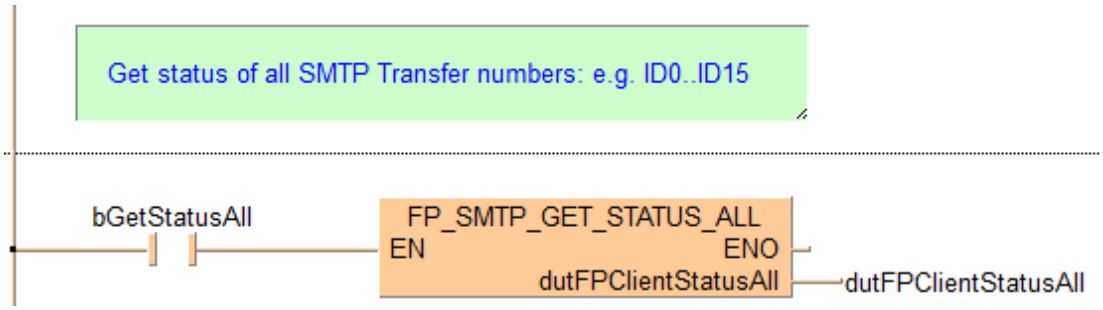
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | dutFPClientStatusAll | FP_CLIENT_STATUS_ALL_DUT | |
| 1 | VAR | bGetStatusAll | BOOL | FALSE |

### POU body

If **bGetStatusAll** is set to TRUE, the instruction is carried out. The status values of all Ethernet units are written into the DUT **dutFPClientStatusAll**.

### LD body



# FP_SMTP_SET_CONNECTION

Server settings for SMTP connection

This FP instruction defines the server settings for the connection to the SMTP client specified in the CPU according to specified parameters.


## Parameters

Input

**sAccount**(STRING)

Set the account data with names and addresses as well as port number and authentication method.

- Source name (optional parameter)
  Keyword: NAME
- Source e-mail address (required parameter)
  Keyword: FROM
  Syntax: `'FROM='`
- IP address or host name (required parameter).
  - IP address
    For an IP address, specify the keyword `IPv4` or `IPv6` at the beginning.
    Syntax for IPv4: e.g. `'IPv4=111.122.133.144'`
    Syntax for IPv6: e.g. `'IPv6=1111:122:2:1555:0:0:1888'`
    Please note that for IPv4 addresses there are range restrictions. When an invalid IP address is specified with an instruction, there will be no operation error but the output **bError** will be set to TRUE.
  - Host name
    Keyword: `HOST`
    Syntax: `'HOST=SMTP.pidsx.com'`
- Port number (optional parameter)
  Keyword: `'PORT'`
  Syntax: `'PORT=xxxxx'` (default: 25)
  Range: 1–65535
- SSL3/TSL1 authentication (optional parameter). Specify whether or not to use SSL3/TSL1 authentication.

Keywords:

- `SSL`: Use SSL3/TLS1
- `NON`: SSL3/TLS1 not used (default)

Examples:

1. Set FP7_001 as the source name, "pana@pana.com" as the source e-mail address, 102.244.2.110 as the IP address and use port 25 with SSL3/TLS1 authentication:
   `'NAME=FP7_001,FROM=pana@pana.com,IPv4=192.255.2.10,PORT=25,SSL'`
2. Keep the source name as is, set "sunx@sunx.com" as the source e-mail address, 1222::a8dd:0:0:6666 as the IP address and use port 100 with SSL3/TLS1 authentication:
   `',FROM=sunx@sunx.com,IPv6=1111:1222::a8dd:0:0:6666,PORT=100,SSL'`
3. Set FP7_002 as the source name, "pewsunx@pewsunx.com" as the source e-mail address, "SMTPmailserver.com" as the host name and use port 1000 without authentication:
   `'NAME=FP7_002,FROM=pewsunx@pewsunx.com,HOST=SMTPmailserver.com,PORT=1000,NON'`
4. Set FP7_002 as the source name, "pewsunx@pewsunx.com" as the source e-mail address, "SMTPmailserver.com" as the host name. Do not change the settings for port number and authentication:
   `'NAME=FP7_002,FROM=pewsunx@pewsunx.com,HOST=SMTPmailserver.com'`

**sLogin**(STRING)

Set the login data

- SMTP authentication method (required keyword)
  Keywords:
  - CRAM: CRAM-MD5 is used
  - PLAIN1: PLAIN1 (ID/PASS) is used
  - PLAIN2: PLAIN2 (ID/PASS) is used
  - LOGIN: LOGIN is used
- Account (max. 32 characters)
  Keyword: ACCOUNT
  Syntax: `'ACCOUNT=xxx'` (default: root). Use `'ACCOUNT='` to delete the account.
- Password (max. 32 characters, upper and lower case characters are allowed)
  Keyword: `PASS`
  Syntax: `'PASS=xxx'` (default: root). Use `'PASS='` to delete the password.

For this parameter, there are two additional keywords available:

- `NOUSE`: The setting for the SMTP authentication method will be ignored.
- `KEEP`: Keeps the current login settings.

Examples:

1. Set CRAM-MD5 as the SMTP authentication method for the account "sunx" with the password "control": `'CRAM,ACCOUNT=sunx,PASS=control'`

2. Set CRAM-MD5 as the SMTP authentication method for the account "root" and delete the password: `'CRAM,ACCOUNT=root,PASS='`
3. Set PLAIN1 as the SMTP authentication method, delete the account name, use the password "SUNX": `'PLAIN1,ACCOUNT=,PASS=SUNX'`
4. Set PLAIN2 as the SMTP authentication method, delete the account name and the password: `'PLAIN2,ACCOUNT=,PASS='`
5. Set LOGIN as the SMTP authentication method for the account "panasonic" and keep the password that has already been set: `'PLAIN2,ACCOUNT=panasonic'`
6. Set CRAM-MD5 as the SMTP authentication method for the account name that has already been specified and set the password "SUNX": `'CRAM,,PASS=SUNX'`
7. Do not use an SMTP authentication method, do not use an account, do not change the password: `'NOUSE'`
8. Keep the SMTP authentication method, the account, and the password: `'KEEP'`

**sCommunication**(STRING)

Set optional communication parameters as required. Note that the default e-mail language is set to Japanese.

- Maximum e-mail size
  Keyword: MAILSIZE
  Syntax: `'MAILSIZE=xxx'` (default: 100)
  Range: 1–10240KB
- Connection timeout
  Keyword: `TOUT`
  Syntax: `'TOUT=xxx'` (default: 60 seconds)
  Range: 30–300 seconds
- Number of retries
  Keyword: `RTRY`
  Syntax: `'RTRY=x'` (default: 3 times)
  Range: 0–3
- Retry interval
  Keyword: `RTTM`
  Syntax: `'RTTM=xxxxx'` (default: 600 seconds)
  Range: 10–86400 seconds
  The value can be specified by 10 seconds. It is rounded down to the 10.
  Example: When you specify 38 seconds, 30 seconds are set.
- Language to be used for e-mail subject and text.
  Keywords:
  – JPN: Japanese (default)
  – ENG: English

For this parameter, there are two additional keywords available:

- `INITIAL`: Resets connection timeout, number of retries, and the retry interval to the default settings.
- `KEEP`: Keeps the current communication settings.

Examples:

1. Set maximum e-mail size: 1000KB, connection timeout: 30 seconds, number of retries: 2, retry interval: 500 seconds, e-mail language: Japanese: `'MAILSIZE=1000,TOUT=30,RTRY=2,RTTM=500,JPN'`
2. Set maximum e-mail size: 10000KB, connection timeout: 270 seconds, no retries, retry interval: 4900 seconds, e-mail language: English: `'MAILSIZE=10000,TOUT=270,RTRY=0,RTTM=4900,ENG'`
3. Set maximum e-mail size: 500KB, connection timeout: 30 seconds, number of retries: 3, retry interval: 200 seconds, e-mail language: do not change: `'MAILSIZE=500,TOUT=30,RTRY=3,RTTM=200'`
4. Set maximum e-mail size: 5000KB, connection timeout: do not change, number of retries: 5, retry interval: 3000 seconds, e-mail language: English: `'MAILSIZE=5000,,RTRY=5,RTTM=3000,ENG'`
5. Reset to default settings (maximum e-mail size: 100KB, connection timeout: 60 seconds, number of retries: 3, retry interval: 600 seconds, default e-mail language: Japanese): `'INITIAL'`
6. Keep all the current settings: `'KEEP'`

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- Separate all keyword entries by comma. e.g. 'NAME=abcd,FROM=sender@server.com'
- The number of characters for string data must not exceed 256.
- This instruction is not available in interrupt programs.
- Upper and lower case characters can be used for specifying keywords. However, the subject, the e-mail text, and the file name of the attachment are case-sensitive.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- The instruction can only be executed when the transfer request flag for the specified transfer setting or the specified **nLogID** number is FALSE. When the transfer request flag is TRUE, an operation error occurs.

- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.
- if the same keyword is specified more than once
- if the e-mail transmission request flag for the specified transfer setting is "TRUE: Transfer requested".
- if the transfer request flag for a specified **nLogID** number is TRUE, e.g. if **sys_bIsLog0DataRecordingActive** is TRUE.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.
- if the same keyword is specified more than once
- if the e-mail transmission request flag for the specified transfer setting is "TRUE: Transfer requested".
- if the transfer request flag for a specified **nLogID** number is TRUE, e.g. if **sys_bIsLog0DataRecordingActive** is TRUE.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed with an incorrect IP address, **sys_iEthernetConnectionErrorCode** is set to "1: Incorrect IP address is specified"
- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bError | BOOL | FALSE |
| 1 | VAR | bSetConnection | BOOL | FALSE |

## POU body

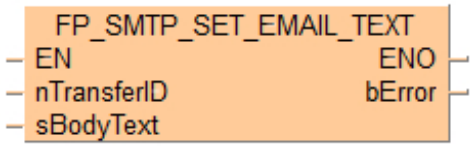
If **bSetConnection** changes from FALSE to TRUE and **sys_bIsEthernetInitializing** is FALSE, the instruction is carried out.

## LD body

# FP_SMTP_SET_EMAIL_TEXT

Set e-mail text

This FP instruction sets the text specified by **sBodyText** as the predefined e-mail text.




## Parameters

Input

**nTransferID** (WORD, INT, UINT)

Set the transfer setting ID.
Values: 0–15

**sBodyText** (STRING)

Stores the text to be used as the predefined e-mail text

Output

**bError** (BOOL)

Set to TRUE when the operation ends abnormally.
When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the e-mail transmission settings using **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client.
- Before you execute the instruction, you need to specify the group and event e-mail settings using **FP_SMTP_SET_GROUP** or the setting dialog of the SMTP client.
- Before you execute the instruction, check **FP_CLIENT_STATUS_DUT** whether there is an active request to send an e-mail. If **bIsTransferRequested** is TRUE when you execute the instruction, an operation error occurs.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.

- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program.
- if the e-mail transmission request flag for the specified transfer setting is "TRUE: Transfer requested".
- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.
- if a destination group number that has not been defined with the instruction **FP_SMTP_SET_GROUP** or the setting dialog of the SMTP client is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program.
- if the e-mail transmission request flag for the specified transfer setting is "TRUE: Transfer requested".
- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.
- if a destination group number that has not been defined with the instruction **FP_SMTP_SET_GROUP** or the setting dialog of the SMTP client is specified.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

### Example

### POU header

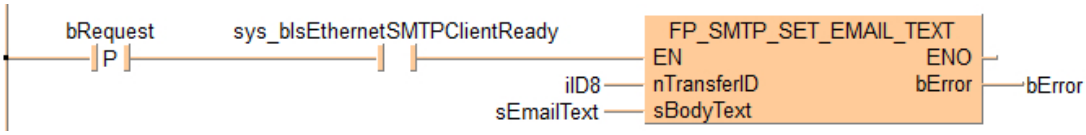
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bRequest | BOOL | FALSE |
| 1 | VAR | bError | BOOL | FALSE |
| 2 | VAR | iID8 | INT | 8 |
| 3 | VAR | sEmailText | STRING[32] | 'Hello World!' |

## POU body

If **bRequest** changes from FALSE to TRUE and **sys_bIsEthernetSMTPClientReady** is set to TRUE, the instruction is executed.

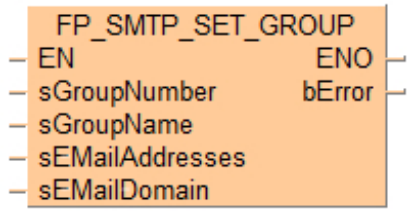
## LD body



# FP_SMTP_SET_GROUP

Set destination group

This FP instruction sets the destination group.




## Parameters

Input

**sGroupNumber** (WORD, INT, UINT)

Set the destination group number.
Values: 0–7
Example: `'GRPNO=3'` specifies group 3 as the destination group.

**sGroupName** (STRING)

Set the destination group name (max. 64 characters).
Example: `'GRPNAME=Group3'` sets "Group3" as the name of the destination group.

**sEMailAddresses** (STRING)

Set the e-mail address of a member belonging to a destination group (max. 256 characters). An e-mail address can be specified with a host name only or host name + domain name. If you omit the domain name here, you must specify it with **sEMailDomain**.
Examples:

1. `'TO=user@support.com'` sets "user@support.com" as the destination e-mail address. If the domain name is included here, you do not need to set a domain name with **sEMailDomain**.
2. `'TO=user'` and `'DOMAIN=support.com'` set "user@support.com" as the destination e-mail address.
3. `'TO=user@support.com,admin@helpcenter.com'` sets two e-mail addresses at two different domains: "user@support.com" and "admin@helpcenter.com".

**sEMailDomain** (STRING)

Set the domain name (max. 32 characters). Only one domain name can be specified.
Setting a domain name allows you to omit the domain name in **sEMailAddresses**, which is helpful if you wish to set multiple e-mail addresses.
Examples:

1. `'TO=user,admin,groupowner'` and `'DOMAIN=support.com'` set multiple e-mail addresses: "user@support.com", "admin@support.com", and "groupowner@support.com".
2. `'TO=user@support.com,admin,groupowner'` and `'DOMAIN=helpcenter.com'` specifies multiple e-mail addresses in different domains: "user@support.com", "admin@helpcenter.com", and "groupowner@helpcenter.com".

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- Separate all keyword entries by comma. e.g. 'NAME=abcd,FROM=sender@server.com'
- The number of characters for string data must not exceed 256.
- This instruction is not available in interrupt programs.
- Upper and lower case characters can be used for specifying keywords. However, the subject, the e-mail text, and the file name of the attachment are case-sensitive.
- Before you execute the instruction, you need to specify the e-mail transmission server using **FP_SMTP_SET_CONNECTION** or the setting dialog of the SMTP client.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- The instruction can only be executed when the transfer request flag for the specified transfer setting or the specified **nLogID** number is FALSE. When the transfer request flag is TRUE, an operation error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program.
- if the e-mail transmission request flag for the specified transfer setting is "TRUE: Transfer requested".

- if the transfer request flag for a specified **nLogID** number is TRUE, e.g. if **sys_blsLog0DataRecordingActive** is TRUE.
- if an e-mail transmission server that has not been specified with the instruction **FP_SMTP_SET_CONNECTION** or the setting dialog of the SMTP client is specified.
- if no domain name has been set with **sEMailDomain** and the e-mail address set with **sEMailAddresses** is specified without domain name.
- if the number of characters for string data exceeds 256.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program.
- if the e-mail transmission request flag for the specified transfer setting is "TRUE: Transfer requested".
- if the transfer request flag for a specified **nLogID** number is TRUE, e.g. if **sys_blsLog0DataRecordingActive** is TRUE.
- if an e-mail transmission server that has not been specified with the instruction **FP_SMTP_SET_CONNECTION** or the setting dialog of the SMTP client is specified.
- if no domain name has been set with **sEMailDomain** and the e-mail address set with **sEMailAddresses** is specified without domain name.
- if the number of characters for string data exceeds 256.

**sys_blsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

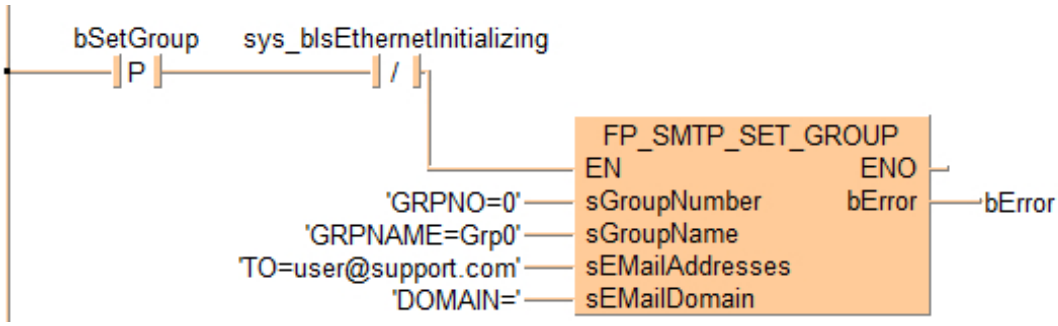
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetGroup | BOOL | FALSE |
| 1 | VAR | bError | BOOL | FALSE |

### POU body

If **bSetGroup** changes from FALSE to TRUE and **sys_blsEthernetInitializing** is FALSE, the instruction is carried out.

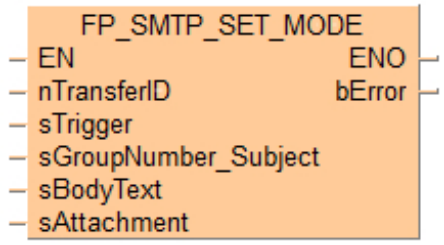
### LD body


bSetGroup
P
sys_bIsEthernetInitializing
/
FP_SMTP_SET_GROUP
EN
ENO
'GRPNO=0'
sGroupNumber
bError
bError
'GRPNAME=Grp0'
sGroupName
'TO=user@support.com'
sEMailAddresses
'DOMAIN='
sEMailDomain

# FP_SMTP_SET_MODE

Set SMTP transfer mode

This FP instruction configures the transmission settings for event-triggered e-mails. Use this instruction together with the instructions **FP_SMTP_CONNECTION** to make the SMTP server settings and **FP_SMTP_SET_GROUP** to specify the destination group.




## Parameters

Input

**nTransferID** (WORD, INT, UINT)

Set the transfer setting ID.
Values: 0–15 (values must be entered in ascending order)

**sTrigger** (STRING)

Set the trigger type with one of the following keyword settings:

- `'TRIG=BITON,xxx'`: Sends an e-mail at the rising edge of the specified bit.

  Values:X, Y, R, L, SR, E, DT.n, LD.n, SD.n

  Example:`'TRIG=BITON,R100'` sends an e-mail at the rising edge of R100.

- `'TRIG=TIME,xxxx,yyyy'`: Sends an e-mail on the specified day and/or at the specified time.

  Values:

  \min,ss (every minute, set time in seconds)

  \hour,mm:ss (every hour, set time in minutes and seconds)

  \day,hh:mm:ss (every day, set time in hours, minutes, and seconds)

  \mon,DD:hh:mm:ss (every month, set date as day, hours, minutes, and seconds)

  \year,MM:DD:hh:mm:ss (every year, set date as month, day, hours, minutes, and seconds)

  \week,hh:mm:ss-w (every week, set day and time as hours, minutes, seconds, and day of week)

  ss=seconds (0–59), mm=minutes (0–59), hh=hours (0–23), DD=day (1–31), MM=month (1–12), w=day of week (0=Sunday, 1=Monday, 2=Tuesday, 3=Wednesday, 4=Thursday, 5=Friday, 6=Saturday)

  Examples:

1. Every day at 13:30: `'TIME,/day,13:30:00'`
2. Every year on April 1st at 9:00: `'TIME,/year,4:1:9:0:0'`
3. Every week on Friday at 23:50: `'TIME,/week,23:50:00-5'`

- `'TRIG=CYCLIC,xxxx'`: Sends an e-mail in regular intervals.

  Values:

  30SEC (30 seconds)

  1MIN, 2MIN, 3MIN, 4MIN, 5MIN, 6MIN, 10MIN, 15MIN, 30MIN (minutes)

  1HOUR, 2HOUR, 3HOUR, 4HOUR, 6HOUR, 12HOUR, 24HOUR (hours)

  The shortest cycle is 30 seconds. You can set only one cycle time (values such as `'1MIN30SEC'` cannot be set).

  Example:`'TRIG=CYCLIC,30SEC'` sends an e-mail every 30 seconds.

- `'TRIG=PROGRAM'`: Sends an e-mail when the instruction **FP_SMTP_TRANSFER_REQUEST** is executed.

- `TRIG=STATUS,xxx1,xxx2,xxx3`: Sends an e-mail when one of the specified PLC status changes occurs.

  Values:

  PROG>RUN (when switching from “PROG” to “RUN”)

  RUN>PROG (when switching from “RUN” to “PROG”)

  ERR>STOP (when a self-diagnostic error occurs and operation stops)

  ERR>RUN (when a self-diagnostic error occurs and operation continues)

  ERRCLR (when an error is cleared)

  Examples:

  1. When the PLC status changes from “PROG” to “RUN”: `'TRIG=STATUS,PROG>RUN'`
  2. When one of the following conditions is met: a self-diagnostic error occurs or when an error is cleared: `'TRIG=STATUS,ERR>STOP,ERR>RUN,ERRCLR'`

**sGroupNumber_Subject**

Set the destination group number and the subject of the e-mail.

- Set up to eight destination group numbers with the keyword syntax `'GRPNO=n1+n2+…+n8'`.

  Values: 0–7

- Set the subject with one of the two following keyword settings:
  - `'SUBJECT=xxxxxx'`: Sets a user-defined subject text.
  - `'SUBJECTAUTO'`: Automatically generated subject containing information about the sending trigger type.

    Automatically generated subjects:

| Japanese | English |
|---|---|
| ビットON 検出（R100） | Bit ON detect (R100) |
| 一定周期メール（ 1 分毎 ） | Interval mail (1 minute) |
| 一定周期メール（24 時間毎 ） | Interval mail (24 hour) |

| Japanese | English |
|---|---|
| 指定時刻メール（毎分・0 秒） | Specified time (every minute 0s) |
| 指定時刻メール（毎時・0 分0 秒） | Specified time (every hour 0m0s) |
| 指定時刻メール（毎日・17 時30 分0 秒） | Specified time (every day 17h30m0s) |
| 指定時刻メール（毎週・金曜・17 時30 分0 秒） | Specified time (every Friday 17h30m00s) |
| PLC 状態変化（電源ON） | PLC status change (power on) |
| PLC 状態変化（PROG > RUN スイッチ切り替え） | PLC status change (PROG > RUN) |
| PLC 状態変化（RUN > PROG スイッチ切り替え） | PLC status change (RUN > PROG) |
| PLC 状態変化（演算停止自己診断エラー検知） | PLC status change (operation stop error) |
| PLC 状態変化（演算継続自己診断エラー検知） | PLC status change (operation continuous error) |
| PLC 状態変化（エラー解除） | PLC status change (error release) |
| SMTPcREQ 命令による | SMRPcREQ command |

**Note:**

- The language used for automatically generated information is specified with the instruction **FP_SMTP_SET_CONNECTION**. Use the parameter **sCommunication** to switch the language from Japanese (default) to English.
- If multiple PLC status changes have been specified as trigger type, the subject will contain the actual status change that triggered the sending of the e-mail.
- The keywords `'GRPNO'` and `'SUBJECT'` must be separated by commas and entered in this order.

Examples:

1. Destination group number 0, user-defined subject "Time notification e-mail":
   `'GRPNO=0,SUBJECT=Time Notification E-Mail'`
2. Destination group numbers 1, 3, 4, 7, user-defined subject "Cyclic notification e-mail":
   `'GRPNO=1+3+4+7,SUBJECT=Cyclic notification e-mail'`
3. Destination group numbers 0 to 7, automatically generated subject:
   `'GRPNO=0+1+2+3+4+5+6+7,SUBJECTAUTO'`

**sBodyText** (STRING)

Specify a variable containing the text of the e-mail or a character string.
Max string length: 4096 one-byte characters for CPUs version 4.1 or later and version 3.4 to 3.x.
256 one-byte characters for all other versions.

**sAttachment** (STRING)

Specify whether or not to send additional information or a file attachment.

Include additional information in the e-mail text with one of the following keyword settings:

- `'INFO=NON'`: Do not add any information automatically
- `'INFO=ADD'`: Add event information (which event has triggered the e-mail)

The following information is added to the e-mail text:

| Japanese | English |
|---|---|
| 基本項目<br>• 送信元<br>• CPU 型番<br>• IPv4 アドレス<br>• IPv6 アドレス | Basic information<br>• From<br>• CPU Part number (e.g. CPS41E)<br>• IPv4 address<br>• IPv6 address |
| トリガ種類 | Detailed information |
| ビットON 検出（R100） | Bit ON detect (R100) |
| 一定周期メール（ 1分毎） | Interval mail (1 minute) |
| 一定周期メール（24 時間毎） | Interval mail (24 hour) |
| 指定時刻メール（毎分・0 秒） | Specified time (every minute 0s) |
| 指定時刻メール（毎時・0 分0 秒） | Specified time (every hour 0m0s) |
| 指定時刻メール（毎日・17 時30 分0 秒） | Specified time (every day 17h30m0s) |
| 指定時刻メール（毎週・金曜・17 時30 分0 秒） | Specified time (every Friday 17h30m00s) |
| PLC 状態変化（電源ON） | PLC status change (power on) |
| PLC 状態変化（PROG＞RUN スイッチ切り替え） | PLC status change (PROG > RUN) |
| PLC 状態変化（RUN＞PROG スイッチ切り替え） | PLC status change (RUN > PROG) |
| PLC 状態変化（演算停止自己診断エラー検知） | PLC status change (operation stop error) |
| PLC 状態変化（演算継続自己診断エラー検知） | PLC status change (operation continuous error) |
| PLC 状態変化（エラー解除） | PLC status change (error release) |
| SMTPcREQ 命令による | SMRPcREQ command |

**Note:**

- The address information depends on whether an IPv4 or IPv6 address has been specified by **FP_SMTP_SET_CONNECTION**.
- The language used for automatically generated information is specified with the instruction **FP_SMTP_SET_CONNECTION**. Use the parameter **sCommunication** to switch the language from Japanese (default) to English.

Include data and/or file attachments with one of the following keyword settings:

- `'ATT=NONE'`: Do not add any data to the e-mail.
- `'ATT=DATA,xxxxxxx'`: Add data from memory area to the e-mail text.

  The following information is added to the e-mail text:

| Information added to e-mail text (Japanese) | Information added to e-mail text (English) |
|---|---|
| デバイス取得項目<br>– デバイス取得（例：DT100）<br>– データ数（例：4 デバイス）<br>– 変換方法<br>– 数値，数値，数値，数値 | Device get information<br>– Device number (e.g. DT100)<br>– Number of data(e.g. 4 registers)<br>– Conversion method<br>– <Numerical value1>, <Numerical value2>, <Numerical value3>, <Numerical value4> |

Specify the start address, the number of data transferred, the conversion method, and the line feed position.

Values:

Memory area and start address: WX, WY, WR, WL, DT, LD, SD

Number of data transferred: 1–1000

Conversion method:

- BIN1w (unconverted 16-bit binary, cannot be used to add data to e-mail texts, use the `'FILE'` keyword instead)
- US (16-bit unsigned decimal)
- SS (16-bit signed decimal)
- UL (32-bit unsigned decimal)
- SL (32-bit signed decimal)
- SF (32-bit single-precision floating point)
- DF (64-bit double-precision floating point)
- HEX1w (16-bit HEX)
- HEX2w (32-bit HEX)
- HEX4w (64-bit HEX)
- ASCII (ASCII character, output is enclosed in "")

Line feed position: 0–255.

- 0: output line feed at the end of the file only.
- n: output line feed after every n data points.

- `'ATT=DATA,xxxxxxxxxxx,FILE=yyyyyyyyyyy'`: Add data from memory area to the e-mail text and additionally attach the data as a file. Use the same values as for `'ATT=DATA,xxxxxxx'`.
- `'ATT=FILE,Filename'`: Attach a file from a specified folder on the SD card.

  Use the keywords `'TOP'` and `'END'` to automatically add date and time to the file name (yymmdd_hhmmss).

  `TOP`: Add data at the beginning of the file name

  `END`: Add data at the end of the file name

**Note:**

- The language used for automatically generated information is specified with the instruction **FP_SMTP_SET_CONNECTION**. Use the parameter **sCommunication** to switch the language from Japanese (default) to English.
- It is not possible to specify LOG folder names for file attachments (`'\LOG0'` to `'\LOG15'`).
- To generate the **sAttachment** string you can also use **FP_SMTP_GET_DATA_FORMAT**.

Examples:

1. Do not add any information or data automatically: `'INFO=NON,ATT=NONE'`
2. Automatically add information, do not add data: `'INFO=ADD,ATT=NONE'`
3. Do not automatically add information, but add 16-bit hex data from DT100 to DT109 to the e-mail text: `'INFO=NON,ATT=DATA,DT100,10,HEX1w'`
4. Do not automatically add information, but attach a file from the SD card: `'INFO=NON,ATT=FIL,\Folder\FileName.bin'`

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- The number of characters for string data should not exceed 4096 for CPU units Ver.3.4 to Ver.3.x and Ver.4.1 or later, and 256 for other CPU units.
- This instruction is not available in interrupt programs.
- Upper and lower case characters can be used for operands for which a character constant can be specified. "Abcd", "ABCD" and "abcd" are synonymous, however, file names are case-sensitive.
- Before you execute the instruction, you need to specify the e-mail transmission server using **FP_SMTP_SET_CONNECTION** or the setting dialog of the SMTP client.
- Before you execute the instruction, you need to specify the group and event e-mail settings using **FP_SMTP_SET_GROUP** or the setting dialog of the SMTP client.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.

- The instruction can only be executed when the transfer request flag for the specified transfer setting or the specified **nLogID** number is FALSE. When the transfer request flag is TRUE, an operation error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.
- if transfer IDs are not specified in ascending order.
- if the instruction is executed in an interrupt program
- if an e-mail transmission server that has not been specified with the instruction **FP_SMTP_SET_CONNECTION** or the setting dialog of the SMTP client is specified.
- if the same destination group number is specified redundantly.
- if a destination group number that has not been defined with the instruction **FP_SMTP_SET_GROUP** or the setting dialog of the SMTP client is specified.
- if the e-mail transmission request flag for the specified transfer setting is "TRUE: Transfer requested".
- if the transfer request flag for a specified **nLogID** number is TRUE, e.g. if **sys_bIsLog0DataRecordingActive** is TRUE.
- if the number of characters for string data exceeds its upper limit. The upper limit is 4096 characters for CPU units Ver.3.4 to Ver.3.x and Ver.4.1 or later, and 256 characters for other CPU units.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.
- if transfer IDs are not specified in ascending order.
- if the instruction is executed in an interrupt program
- if an e-mail transmission server that has not been specified with the instruction **FP_SMTP_SET_CONNECTION** or the setting dialog of the SMTP client is specified.
- if the same destination group number is specified redundantly.
- if a destination group number that has not been defined with the instruction **FP_SMTP_SET_GROUP** or the setting dialog of the SMTP client is specified.
- if the e-mail transmission request flag for the specified transfer setting is "TRUE: Transfer requested".
- if the transfer request flag for a specified **nLogID** number is TRUE, e.g. if **sys_bIsLog0DataRecordingActive** is TRUE.

- if the number of characters for string data exceeds its upper limit. The upper limit is 4096 characters for CPU units Ver.3.4 to Ver.3.x and Ver.4.1 or later, and 256 characters for other CPU units.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

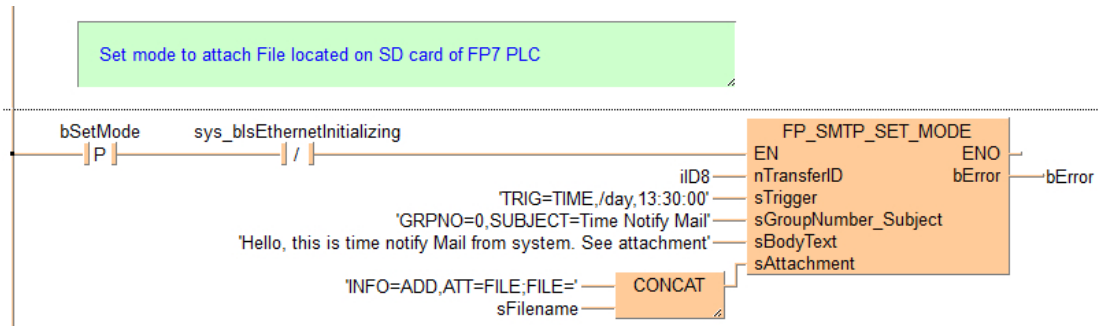
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetModePlcData | BOOL | FALSE |
| 1 | VAR | bSetModePlcDataError | BOOL | FALSE |
| 2 | VAR | arRealArray | ARRAY [0..15] OF REAL | [16(0.0)] |
| 3 | VAR | iID10 | INT | 10 |

### POU body

If **bSetModePLCData** changes from FALSE to TRUE and **sys_bIsEthernetInitializing** is FALSE, the instruction is carried out.

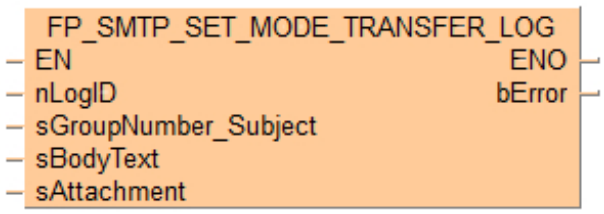
### LD body



# FP_SMTP_SET_MODE_TRANSFER_LOG

Set SMTP transfer mode for data recording files

This FP instruction defines the SMTP client transfer settings for e-mail when the data recording file has been completed.




## Parameters

Input

**nLogID** (WORD, INT, UINT)

LOG number (permissible range: 0–15). The compiler generates internally the string for the log file number, e.g. 'LOG=0'

**sGroupNumber_Subject** (STRING)

Set the destination group number and the subject of the e-mail.

- Set up to eight destination group numbers with the keyword syntax `'GRPNO=n1+n2+…+n8'`.

  Values: 0–7

- Set the subject with one of the two following keyword settings (max. 64 characters, one-byte):
  - `'SUBJECT=xxxxxx'`: Sets a user-defined subject text.
  - `'SUBJECTAUTO'`: Automatically generated subject containing information about the sending trigger type.

    Automatically generated subjects:

| Japanese | English |
|---|---|
| ロギング / トレース (LOG0) | Logging/Trace (LOG0) |
| ロギング / トレース (LOG1) | Logging/Trace (LOG1) |
| ... | ... |
| ロギング / トレース (LOG14) | Logging/Trace (LOG14) |
| ロギング / トレース (LOG15) | Logging/Trace (LOG15) |

**Note:**

– The language used for automatically generated information is specified with the instruction **FP_SMTP_SET_CONNECTION**. Use the parameter **sCommunication** to switch the language from Japanese (default) to English.

– The keywords `'GRPNO'` and `'SUBJECT'` must be separated by commas and entered in this order.

Examples:

1. Destination group number 0, user-defined subject "LogFileTransmission":
   `'GRPNO=0,SUBJECT=LogFileTransmission'`
2. Destination group numbers 1, 3, 4, 7, user-defined subject "LogFilesGroups":
   `'GRPNO=1+3+4+7,SUBJECT=LogFilesGroups'`
3. Destination group numbers 0 to 7, automatically generated subject:
   `'GRPNO=0+1+2+3+4+5+6+7,SUBJECTAUTO'`

**sBodyText** (STRING)

Message (max. 256 characters, one byte)
Specify the variable storing e-mail text or a character constant.

**sAttachment** (STRING)

Specify whether or not to send additional information or a file attachment.
Include additional information in the e-mail text with one of the following keyword settings:

- `'INFO=NONE'`: Do not add any information automatically
- `'INFO=AUTO'`: Add automatically generated text

The following information is added to the e-mail text:

| Japanese | English |
|---|---|
| 基本項目<br>• 送信元<br>• CPU 型番 ( 例 : CPS41E )<br>• IPv4 アドレス<br>• IPv6 アドレス<br>• ロギングトレース(ID 番号)<br>• ファイル確定時刻 | Basic information<br>• From<br>• CPU Part number (e.g. CPS41E)<br>• IPv4 address<br>• IPv6 address<br>• Logging Trace ID<br>• File fixed Time |

**Note:**

- The language used for automatically generated information is specified with the instruction **FP_SMTP_SET_CONNECTION**. Use the parameter **sCommunication** to switch the language from Japanese (default) to English.
- To generate the **sAttachment** string you can also use **FP_SMTP_GET_DATA_FORMAT**.

Attach the data recording file (LOG file) with the following keyword setting:

- `'ATT=NONE'`: Do not attach the LOG file to the e-mail.
- `'ATT=FILE'`: Attach the LOG file with the number specified by **nLogID**.

Examples:

1. Do not add any information, do not attach the LOG file: `'INFO=NONE,ATT=NONE'`
2. Add automatically generated information, attach the LOG file: `'INFO=ADD,ATT=FILE'`

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:
- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- The parameters for the automatic text generation and file attachment cannot be omitted.
- Separate all keyword entries by comma. e.g. 'NAME=abcd,FROM=sender@server.com'
- Do not change the order of keywords. Specify the keywords and their setting parameters in the order they are listed here.
- The number of characters for string data must not exceed 256.
- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the e-mail transmission server using **FP_SMTP_SET_CONNECTION** or the setting dialog of the SMTP client.
- Before you execute the instruction, you need to specify the group and event e-mail settings using **FP_SMTP_SET_GROUP** or the setting dialog of the SMTP client.
- Upper and lower case characters can be used for specifying keywords. However, the subject, the e-mail text, and the file name of the attachment are case-sensitive.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- The instruction can only be executed when the transfer request flag for the specified **nLogID** number is FALSE. When the transfer request flag is TRUE, an operation error occurs.
- The instruction stores the e-mail settings for data recording of **sGroupNumber_Subject**, **sBodyText** and **sAttachment** in the LOG file number specified by **nLogID**.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the transfer request flag for a specified **nLogID** number is TRUE, e.g. if **sys_bIsLog0DataRecordingActive** is TRUE.
- if the data recording condition of a specified **nLogID** number is not registered.
- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if a destination group number that has not been defined with the instruction **FP_SMTP_SET_GROUP** or the setting dialog of the SMTP client is specified.
- if no e-mail transmission server is specified.
- if an e-mail transmission server that has not been specified with the instruction **FP_SMTP_SET_CONNECTION** or the setting dialog of the SMTP client is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the transfer request flag for a specified **nLogID** number is TRUE, e.g. if **sys_bIsLog0DataRecordingActive** is TRUE.
- if the data recording condition of a specified **nLogID** number is not registered.
- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.
- if a destination group number that has not been defined with the instruction **FP_SMTP_SET_GROUP** or the setting dialog of the SMTP client is specified.
- if no e-mail transmission server is specified.
- if an e-mail transmission server that has not been specified with the instruction **FP_SMTP_SET_CONNECTION** or the setting dialog of the SMTP client is specified.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

## POU header

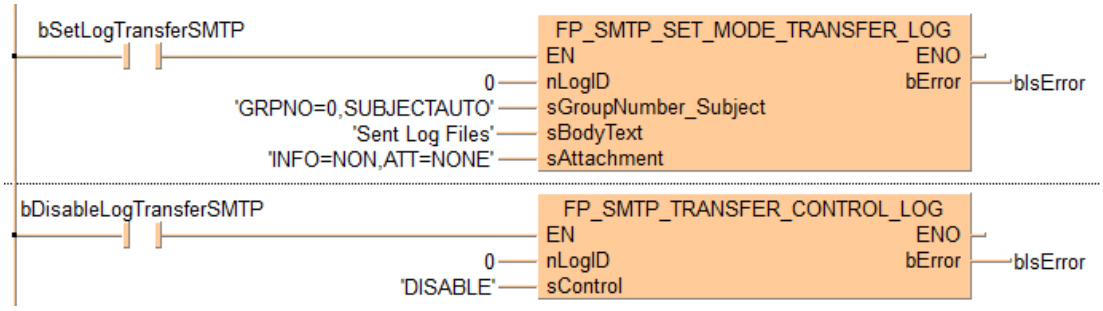
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bIsError | BOOL | FALSE |
| 1 | VAR | bSetLogTransferSMTP | BOOL | FALSE |
| 2 | VAR | bDisableLogTransferSMTP | BOOL | FALSE |

## POU body

If **bSetLogTransferSMTP** and **bDisableLogTransferSMTP** are set to TRUE, the instruction is carried out.

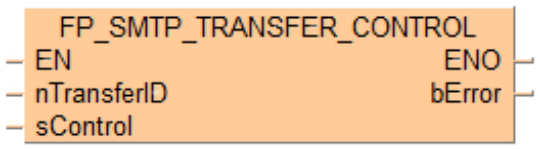
## LD body



# FP_SMTP_TRANSFER_CONTROL

Control of SMTP transfer to a single Ethernet unit

This FP instruction controls the SMTP transfer to a single Ethernet unit specified by **nTransferID**. Valid control words are `'ENABLE'`, `'DISABLE'` and `'CANCEL'`.




## Parameters

Input

**nTransferID** (INT)

Ethernet unit ID (values: 0–15)

**sControl** (STRING)

Control string:

- `'ENABLE'`: enables e-mail transfer to the Ethernet unit
- `'DISABLE'`: disables e-mail transfer to the Ethernet unit
- `'CANCEL'`: cancels e-mail transfer to the Ethernet unit

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- The number of characters for string data must not exceed 256.
- Upper and lower case characters can be used for operands for which a character constant can be specified. "Abcd", "ABCD" and "abcd" are synonymous, however, file names are case-sensitive.
- Before you execute the instruction, you need to specify the e-mail transmission settings using **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client.

- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- It takes some time to process the transfer cancel request. Check the transfer status with **FP_SMTP_GET_STATUS** and check if the transfer stops after executing the instruction.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the number of characters for string data exceeds 256.
- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the number of characters for string data exceeds 256.
- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

## POU header

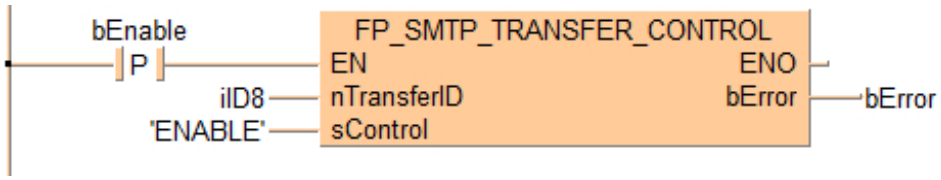
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | bError | BOOL | FALSE |
| 2 | VAR | iID8 | INT | 8 |

### POU body

If **bEnable** changes from **FALSE** to TRUE, the instruction is carried out. The SMTP transfer is enabled for Ethernet unit 8.

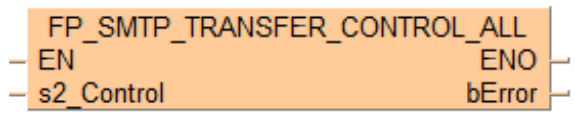
### LD body



# FP_SMTP_TRANSFER_CONTROL_ALL

Control of SMTP transfer to all Ethernet units

This FP instruction controls the SMTP transfer to all Ethernet units. Valid control words are `'ENABLE'`, `'DISABLE'` and `'CANCEL'`.




## Parameters

Input

**s2_Control** (STRING)

Control string:

- `'ENABLE'`: enables e-mail transfer to all Ethernet units
- `'DISABLE'`: disables e-mail transfer to all Ethernet units
- `'CANCEL'`: cancels e-mail transfer to all Ethernet units

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- The number of characters for string data must not exceed 256.
- Upper and lower case characters can be used for operands for which a character constant can be specified. "Abcd", "ABCD" and "abcd" are synonymous, however, file names are case-sensitive.
- Before you execute the instruction, you need to specify the e-mail transmission settings using **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.

- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- It takes some time to process the transfer cancel request. Check the transfer status with **FP_SMTP_GET_STATUS** and check if the transfer stops after executing the instruction.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the number of characters for string data exceeds 256.
- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the number of characters for string data exceeds 256.
- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

### Example

### POU header

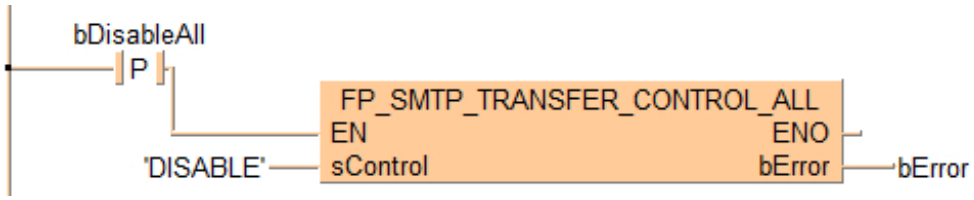
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bDisableAll | BOOL | FALSE |
| 1 | VAR | bError | BOOL | FALSE |

### POU body

If **bDisableAll** changes from FALSE to TRUE, the instruction is carried out. The SMTP transfer is disabled for all Ethernet units.

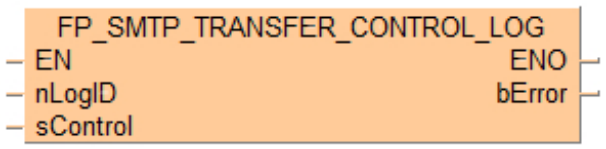
### LD body


bDisableAll
P
FP_SMTP_TRANSFER_CONTROL_ALL
EN
ENO
'DISABLE'
sControl
bError
bError

# FP_SMTP_TRANSFER_CONTROL_LOG

Control of SMTP transfer of data recording files

This FP instruction controls the SMTP transfer of a data recording file specified by **nLogID**. Valid control words are `'ENABLE'`, `'DISABLE'` and `'CANCEL'`.




## Parameters

Input

**nLogID** (WORD, INT, UINT)

LOG number (permissible range: 0–15). The compiler generates internally the string for the log file number, e.g. 'LOG=0'

**sControl** (STRING)

Control string:

- `'ENABLE'`: enables file transfer
- `'DISABLE'`: disables file transfer
- `'CANCEL'`: cancels file transfer

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- The number of characters for string data must not exceed 256.
- Before you execute the instruction, you need to specify the e-mail transfer settings for data recording files using **FP_SMTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the SMTP client.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed.

When you execute the instruction while **sys_blsEthernetInitializing** is TRUE, an error occurs.

- Upper and lower case characters can be used for operands for which a character constant can be specified. "Abcd", "ABCD" and "abcd" are synonymous, however, file names are case-sensitive.
- When this instruction has been executed successfully, the system variables **sys_blsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- It takes some time to process the transfer cancel request. Check the transfer status with **FP_SMTP_GET_STATUS** and check if the transfer stops after executing the instruction.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the number of characters for string data exceeds 256.
- if an e-mail transmission setting for data recording files that has not been configured with the instruction **FP_SMTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the SMTP client is specified.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if any control string other than `'ENABLE'`, `'DISABLE'` or `'CANCEL'` is specified for **sControl**.
- if the number of characters for string data exceeds 256.
- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.
- if an e-mail transmission setting for data recording files that has not been configured with the instruction **FP_SMTP_SET_MODE_TRANSFER_LOG** or the setting dialog of the SMTP client is specified.

**sys_blsCarry** (turns to TRUE for one scan)

- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

## POU header

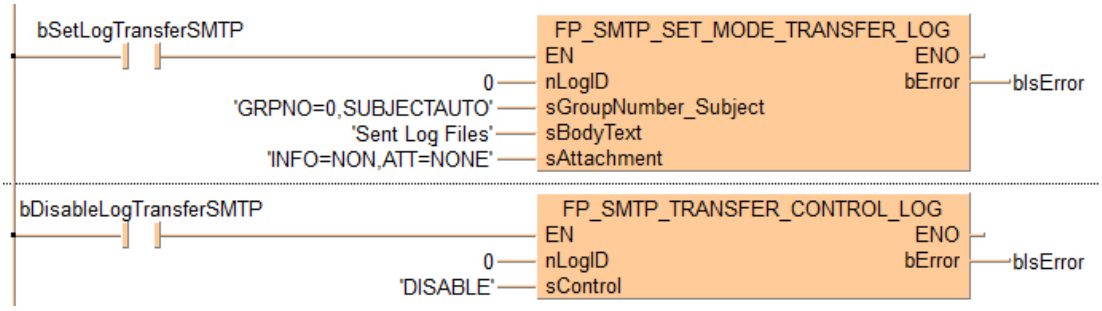
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bIsError | BOOL | FALSE |
| 1 | VAR | bSetLogTransferSMTP | BOOL | FALSE |
| 2 | VAR | bDisableLogTransferSMTP | BOOL | FALSE |

## POU body

If **bSetLogTransferSMTP** and **bDisableLogTransferSMTP** are set to TRUE, the instruction is carried out.

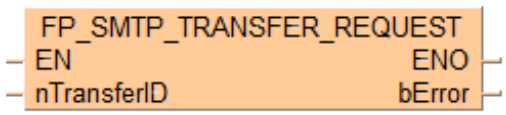
## LD body



# FP_SMTP_TRANSFER_REQUEST

SMTP transfer request

This FP instruction requests the SMTP client to send an e-mail and transfer data as specified by **nTransferID**.




## Parameters

Input

**nTransferID** (WORD, INT, UINT)

Set the SMTP client ID
Values: 0–15

Output

**bError** (BOOL)

Turns to TRUE under the following conditions:

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- This instruction is not available in interrupt programs.
- Before you execute the instruction, you need to specify the e-mail transmission settings using **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client.
- Before you execute the instruction, check if the system variable **sys_bIsEthernetSMTPClientReady** is TRUE. If it is FALSE when executing the instruction, an error occurs.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.
- The instruction can only be executed when **sys_bIsEthernetCableNotConnected** is FALSE.

- The instruction can only be executed when the transfer request flag for the specified transfer setting is "FALSE: No transfer request". When the transfer request flag is "TRUE: Transfer requested", an operation error occurs.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program.
- if the system variable **sys_bIsEthernetSMTPClientReady** is FALSE when the instruction is executed.
- if the e-mail transmission request flag for the specified transfer setting is "TRUE: Transfer requested".
- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.
- if sending of e-mails is disabled, i.e. if the identifier **bIsTransmissionBlocked** of **FP_CLIENT_STATUS_DUT** is TRUE.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if the instruction is executed in an interrupt program.
- if the system variable **sys_bIsEthernetSMTPClientReady** is FALSE when the instruction is executed.
- if the e-mail transmission request flag for the specified transfer setting is "TRUE: Transfer requested".
- if an e-mail transmission setting that has not been configured with the instruction **FP_SMTP_SET_MODE** or the setting dialog of the SMTP client is specified.
- if sending of e-mails is disabled, i.e. if the identifier **bIsTransmissionBlocked** of **FP_CLIENT_STATUS_DUT** is TRUE.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the instruction is executed while the Ethernet cable is disconnected. **sys_iEthernetConnectionErrorCode** is set to "10: Ethernet cable disconnected".
- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

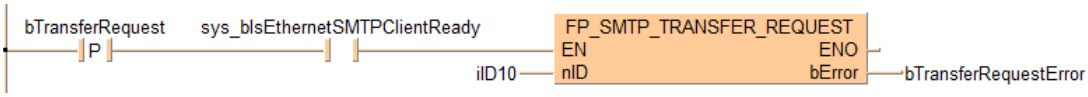
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iID10 | INT | 10 |
| 1 | VAR | bTransferRequestError | BOOL | FALSE |
| 2 | VAR | bTransferRequest | BOOL | FALSE |

### POU body

If **bTransferRequest** changes from **FALSE** to TRUE and the system variable **sys_bIsEthernetSMTPClientReady** is TRUE, the instruction is carried out. The SMTP transfer is requested for the Ethernet unit 10.

### LD body



# 7 Communication instructions for Modbus/MEWTOCOL

## 7.1 Introduction to communication instructions for Modbus/MEWTOCOL

This communication mode uses the MEWTOCOL-COM or Modbus RTU protocol to exchange data between a master and one or more slaves. This is called 1:1 or 1:N communication. The side that issues commands is called master. The slave receives the commands, executes the process and sends back responses. The slave answers automatically to the commands received from the master, so no program is necessary on the slave.


(1) Command message
(2) Response message

### Note

The MEWTOCOL-COM master function is not supported by all PLCs.

### Modbus specifications for Panasonic PLCs:

| Modbus area address | Modbus memory name | Memory type | Panasonic PLCs address |
|---|---|---|---|
| 000001 … | COIL | 1-bit | Y0 … |
| 002049 … | | | R0 … |
| 100001 … | INPUT | 1-bit | X0 … |
| 400001 … | HOLDING_REGISTER | 16-bit | DT0 … |
| 300001 … | INPUT_REGISTERS | 16-bit | WL0 … |
| 302001 … | | | LD0 … |

For reference number and address area ranges supported by the Panasonic PLCs, please refer to the User's Manual of the PLC. If the reference number is outside the supported range, an error is returned.

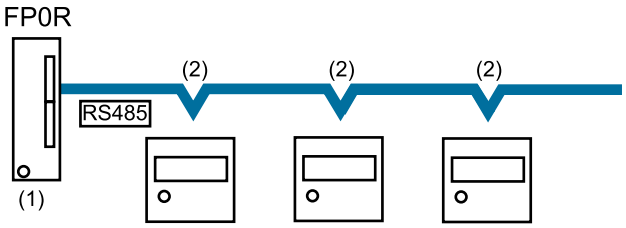
The reference numbers are part of the Modbus standard and are required any time other devices are connected to Panasonic PLCs with the aim of communicating via Modbus.

### Related topics

FP-X0, MCU: For PDF files, please refer to Panasonic Download Center

## 7.2 Master function

Write and read access to various slaves is possible using the F145 and F146 instructions. Individual access to each slave as well as global transmission is possible.




(1) Master
(2) Slave

### Procedure for master communication

1. Set communication parameters

   Required settings: communication mode (MEWTOCOL-COM or Modbus RTU), station number, baud rate, communication format

2. Execute write or read instruction

   Use one of the following instructions:

| Instruction | Application | Restrictions |
|---|---|---|
| **F145_WRITE_DATA** | Easy to use for communication with Panasonic devices supporting the MEWTOCOL-COM or Modbus RTU slave function | Devices must have matching address areas |
| **F146_READ_DATA** | | |
| **F145_WRITE_DATA_TYPE_OFFS** | | – |
| **F146_READ_DATA_TYPE_OFFS** | | |
| **F145F146_MODBUS_MASTER** | For communication with any device supporting the Modbus RTU protocol | For station numbers 1–99 only, start register range of 0–32764<br>Start register range of 0–65535 |

3. Evaluate flags

   Use one of the following instructions:

| Method | Comment |
|---|---|
| **IsF145F146NotActive** | Returns the value of the "F145/F146 not active" flag. It is TRUE if the execution of an F145 or F146 instruction is possible because neither instruction is active. |
| **sys_bIsComPort1F145F146NotActive**<br>**sys_bIsComPort2F145F146NotActive** | These system variables are TRUE if the execution of an F145 or F146 instruction is possible because neither instruction is active. |
| **IsF145F146Error** | Returns the value of the "F145/F146 error" flag. It is TRUE if the execution of an F145 or F146 instruction has terminated with an error. |
| **sys_bIsComPort1F145F146Error**<br>**sys_bIsComPort2F145F146Error** | These system variables are TRUE if the execution of an F145 or F146 instruction has terminated with an error. |
| **sys_wComPort1F145F146ErrorCode**<br>**sys_bIsComPort2F145F146ErrorCode** | These system variables contain the error code if transmission has terminated with an error. |

Note

The optional user library **NCL-MODBUS-LIB** for Control FPWIN Pro offers the most complete Modbus functionality, including function blocks for multi-master and multi-slave applications. The library can be used with the FPΣ Serial Data Unit and the FP2/FP2SH Multi-Communication Unit.

Related topics

Setting communication parameters (page 529)

**F145_WRITE_DATA**

**F146_READ_DATA**

**F145_WRITE_DATA_TYPE_OFFS**

**F146_READ_DATA_TYPE_OFFS**

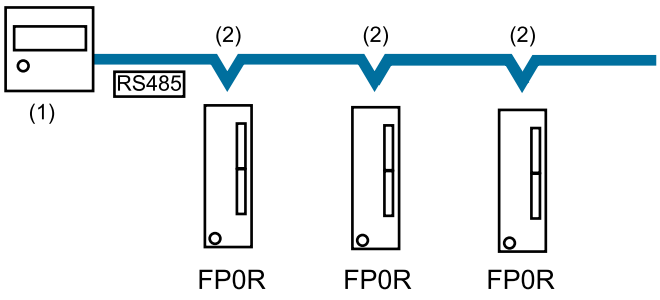
**F145F146_MODBUS_MASTER**

**IsF145F146NotActive**

**IsF145F146Error**

## 7.3 Slave function

After having received a command message from the master station, the slave stations send back the response message based on the instructions received. Do not execute the F145_WRITE and F146_READ instructions on slave stations.




(1) Master
(2) Slave

Slave communication

1. Setting communication parameters
   Required settings: communication mode (MEWTOCOL-COM or Modbus RTU), station number, baud rate, communication format
2. Receive data from the master
   There is no program required on the slave. The program for the master side must send and receive commands according to the MEWTOCOL-COM or Modbus-RTU protocol. The protocols contain the commands used to control and monitor the slave operation.

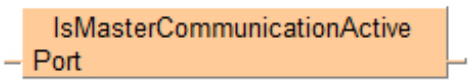
## Related topics

Setting communication parameters (page 529)

# IsMasterCommunicationActive

Evaluate "IsMasterCommunicationActive" flag

This instruction returns the value of the "IsMasterCommunicationActive" flag.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

## Remarks

The flag "IsMasterCommunicationActive" can be evaluated using one of the following system variables:

- **sys_bIsComPort0MasterCommunicationActive**
- **sys_bIsComPort1MasterCommunicationActive**
- **sys_bIsComPort2MasterCommunicationActive**

For communication ports not supporting the master function, the flag is always TRUE.

# Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iPort | INT | 0 |
| 1 | VAR | bIsMasterCommunicationActive | BOOL | FALSE |

### LD body


IsMasterCommunicationActive
iPort
Port
bIsMasterCommunicationActive

## 7.5 FP instructions

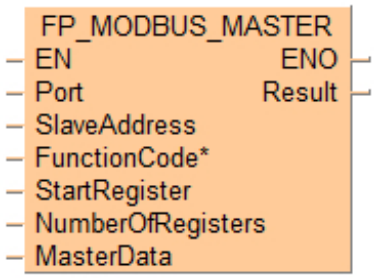
Tip

Advantages of FP instructions

# FP_MODBUS_MASTER

Write data to slave or read data from slave

Use this instruction to write data from a master to a slave or read data from a slave via the communication port using the Modbus RTU protocol, as defined in the system registers of the port used. Make sure the same protocol is set for master and slave.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**SlaveAddress** (WORD, INT, UINT)

Station number of the slave

- For FP7 only:
  - Function codes 05, 06, 15, 16: 0–247

    When SCU is used and "0" is specified for the slave unit number, a global transfer is executed. In this case, there is no response message from the slave unit.
  - Function codes 01, 02, 03, 04: 1-247
- For other PLCs:
  - Function codes 05, 06, 15, 16: 0–99

    When "0" is specified for the slave unit number, a global transfer is executed. In this case, there is no response message from the slave unit.
  - Function codes 01, 02, 03, 04: 1-99

Set to 1, if a **SYS_ETHERNET_USER_CONNECTION_xx** is applied to input **Port**

**FunctionCode*** (WORD, INT, UINT)

**SYS_MODBUS_01_READ_COIL** (page 381)

**SYS_MODBUS_02_READ_INPUT** (page 381)
**SYS_MODBUS_03_READ_HOLDING_REGISTER** (page 382)
**SYS_MODBUS_04_READ_INPUT_REGISTERS** (page 382)
**SYS_MODBUS_05_FORCE_COIL** (page 383)
**SYS_MODBUS_06_PRESET_REGISTER** (page 384)
**SYS_MODBUS_15_FORCE_COILS** (page 384)
**SYS_MODBUS_16_PRESET_REGISTERS** (page 385)

**StartRegister** (WORD, INT, UINT)

Starting address (0–65535). The address type depends on the command specified by **FunctionCode***.

**NumberOfRegisters*** (WORD, INT, UINT)

Number of transmission bits or words.

- 1 for function codes 05, 06
- 1–2040 for function codes 01, 02
- 2–2040 for function code 15
- 1–127 for function codes 03, 04
- 2–127 for function code 16

**MasterData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

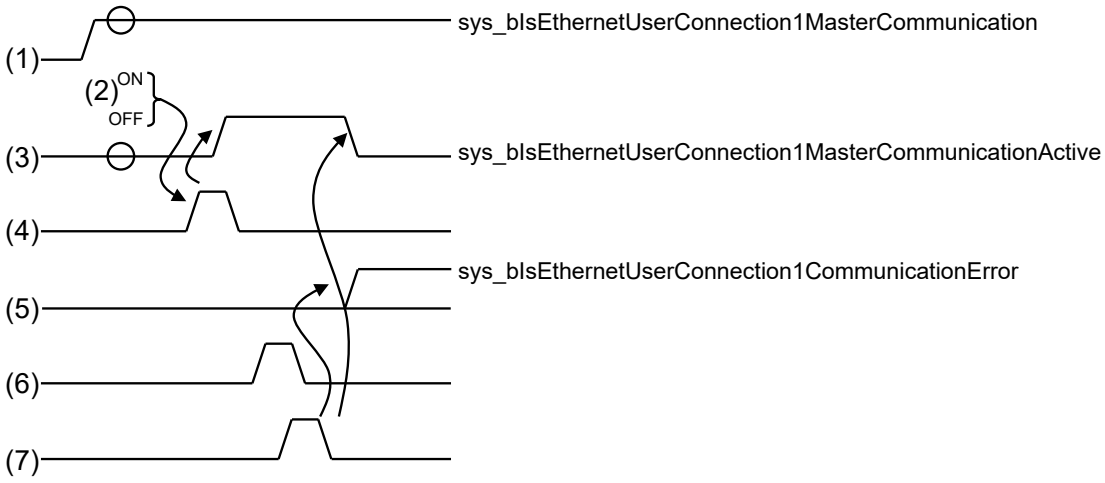
Source address on the master for the data to be written to the slave.

Output

**Result** (WORD, INT, UINT)

For FP7 only: For a description of all error codes, please refer to the table of Modbus/MEWTOCOL communication error codes.
For other PLCs: set to 0

## Time chart




(1) Master communication clear-to-send flag,e.g.
**sys_bIsEthernetUserConnection1MasterCommunication**

(2) Check that the master communication clear-to-send flag is TRUE and check that the master communication sending flag is FALSE

(3) Master communication sending flag, e.g.
**sys_bIsEthernetUserConnection1MasterCommunicationActive**
While sending: Master communication sending flag is TRUE
Sending done: Master communication sending flag is FALSE

(4) Execute this instruction

(5) Master communication sending done flag, e.g.
**sys_bIsEthernetUserConnection1CommunicationError**
Normal completion: FALSE
Abnormal completion: TRUE

(6) Send data

(7) Processing of the received response

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- If slave or master data exceed the available address range.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- If slave or master data exceed the available address range.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

## Example

### SYS_MODBUS_01_READ_COIL

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

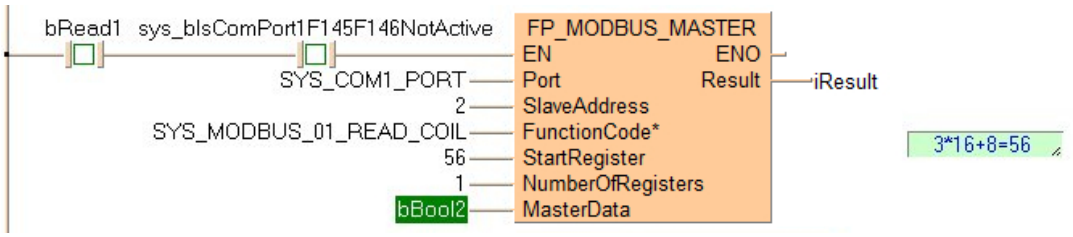
For the special case that the connected slave is a Panasonic PLC configured as Modbus RTU slave via system register, one or multiple bits are read from:

- Y (Output)
- R (Internal flags)

#### Example

Executing Modbus command 01: reads 1 bit from a Modbus slave beginning at start register 56 set by the variable **StartRegister**. Then the command stores the 1 bit in the master beginning at the address set by the variable **bBool2**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, the start register 56 corresponds to output **Y38** (3*16+8=56).




### SYS_MODBUS_02_READ_INPUT

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

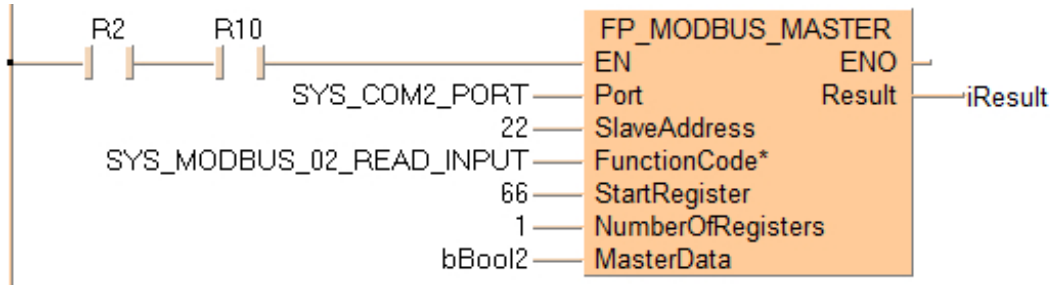
For the special case that the connected slave is a Panasonic PLC configured as Modbus RTU slave via system register, one or multiple bits are read from:

- X (Input)

#### Example

Executing Modbus command 02: reads 1 bit from a Modbus slave beginning at start register 66 set by the variable **StartRegister**. Then the command stores the 1 bit in the master beginning at the address set by the variable **bBool2**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, start register 66 corresponds to input **X42** (4*16+2=66).




## SYS_MODBUS_03_READ_HOLDING_REGISTER

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

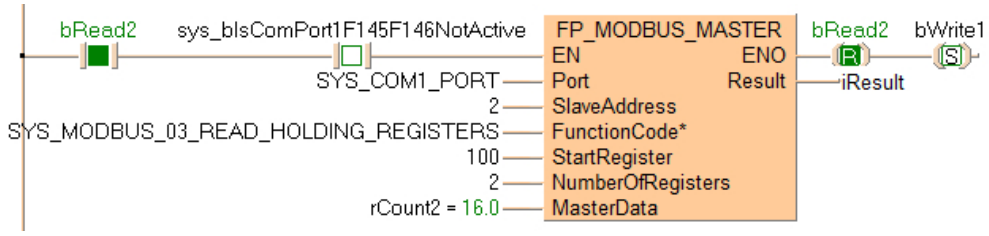
For the special case that the connected slave is a Panasonic PLC configured as Modbus RTU slave via system register, one or multiple bits are read from:

- DT (data registers)

### Example

Executing Modbus command 03: reads 2 words from a Modbus slave beginning at start register 100 set by the variable **StartRegister**. Then the command stores the 2 words in the Modbus master 2 beginning at the address set by the variable **rCount2**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, start register 100 corresponds to data register DT100.




Maximum number of registers: 127.

## SYS_MODBUS_04_READ_INPUT_REGISTERS

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

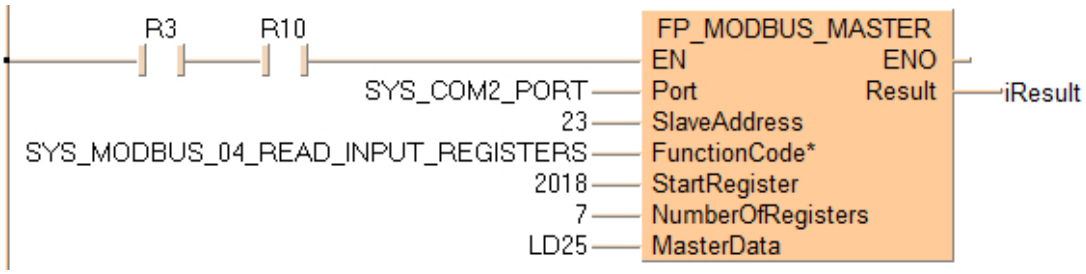
For the special case that the connected slave is a Panasonic PLC configured as Modbus RTU slave via system register, one or multiple bits are read from:

- WL0–WL127 (Link flags)
- LD0–LD256 (Link registers)

### Example

Executing Modbus command 04: reads 7 words from a Modbus slave beginning at start register 2018 set by the variable **StartRegister**. Then the command stores the 7 words in the master beginning at LD25 set by the variable **MasterData**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, start register 2018 corresponds to link register LD18.




Maximum number of registers: 127.

## SYS_MODBUS_05_FORCE_COIL

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

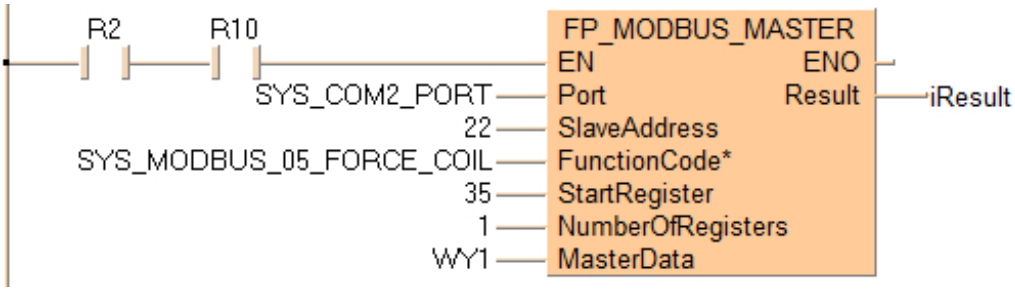
In case the connected slave is a Panasonic PLC in Modbus RTU mode, one or multiple bits are written to:

- Y (Output)
- R (Internal flags)

### Example

Executing Modbus command 05: writes 1 bit to a Modbus slave beginning at address **WY1** set by the variable **MasterData**. Then the command stores the 1 bit in a Modbus slave beginning at start register 35 set by the variable **StartRegister**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, address 35 corresponds to output **Y23** (2*16+3=35).




When writing multiple bit data use the SYS_MODBUS_15_FORCE_COILS constant.

Maximum number of register: 1

### SYS_MODBUS_06_PRESET_REGISTER

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

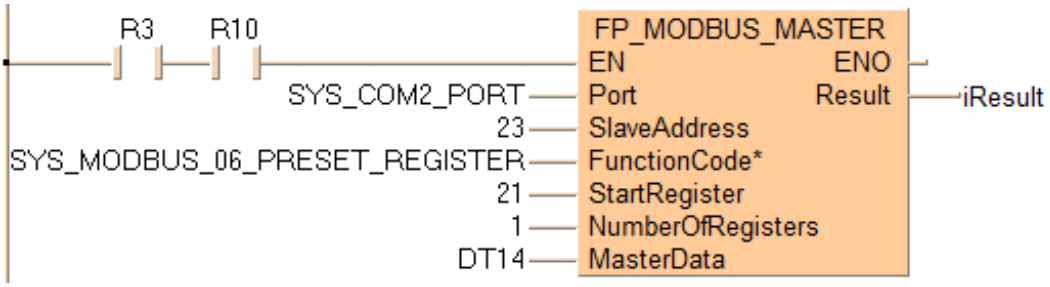
In case the connected slave is a Panasonic PLC in Modbus RTU mode, one or multiple bits are written to:

- DT (data registers)

#### Example

Executing Modbus command 06: writes 1 word to a Modbus slave beginning at address DT14 set by the variable **MasterData**. Then the command stores the 1 word in the Modbus slave beginning at start register 21 set by the variable **StartRegister**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, start register 21 corresponds to data register DT21.




When writing multiple bit data use the SYS_MODBUS_15_FORCE_COILS constant.

Maximum number of register: 1

### SYS_MODBUS_15_FORCE_COILS

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

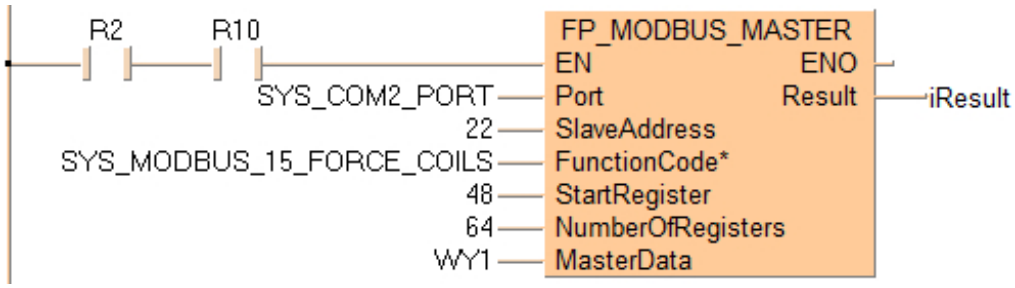
In case the connected slave is a Panasonic PLC in Modbus RTU mode, one or multiple bits are written to:

- Y (Output)
- R (Internal flags)

#### Example

Executing Modbus command 15: writes 64 bits to a Modbus slave beginning at address **WY1** set by the variable **MasterData**. Then the command stores the 64 bits in the Modbus slave beginning at start register 48 set by the variable **StartRegister**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, address 48 corresponds to output **Y30** (3*16=48).




## SYS_MODBUS_16_PRESET_REGISTERS

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

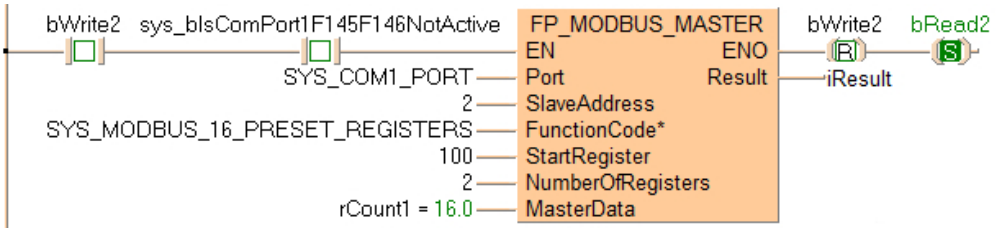
In case the connected slave is a Panasonic PLC in Modbus RTU mode, one or multiple bits are written to:

- DT (data registers)

### Example

Executing Modbus command 16: writes data to a Modbus slave beginning at the address set by the variable **rCount1**. Then the command stores the data in the Modbus slave 2 beginning with start register 100 set by the variable **StartRegister**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, start register 100 corresponds to data register DT100.



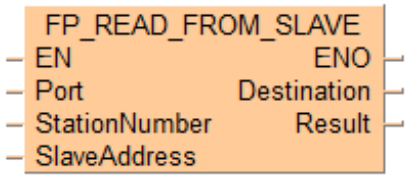

Maximum number of registers: 127.

# FP_READ_FROM_SLAVE

Read data from slave

Use this instruction to request data from a slave via the communication port using the MEWTOCOL-COM or Modbus RTU protocol, as defined in the system registers of the port used. Make sure the same protocol is set for master and slave. Master and slave must have matching memory areas. If the slave data is not available in the user area of the master, use **FP_READ_FROM_SLAVE_AREA_OFFS** or **FP_MODBUS_MASTER**.

For data transmissions using the Modbus protocol, the compiler generates Modbus commands based on the Modbus reference numbers.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**StationNumber** (WORD, INT, UINT)

Station number of the slave (MEWTOCOL: 1–99, MODBUS: 1–247)
Set to 1, if a **SYS_ETHERNET_USER_CONNECTION_xx** is applied to input **Port**

**SlaveAddress** (WORD, INT, UINT)

Source address on the slave from which the data is requested.

Output

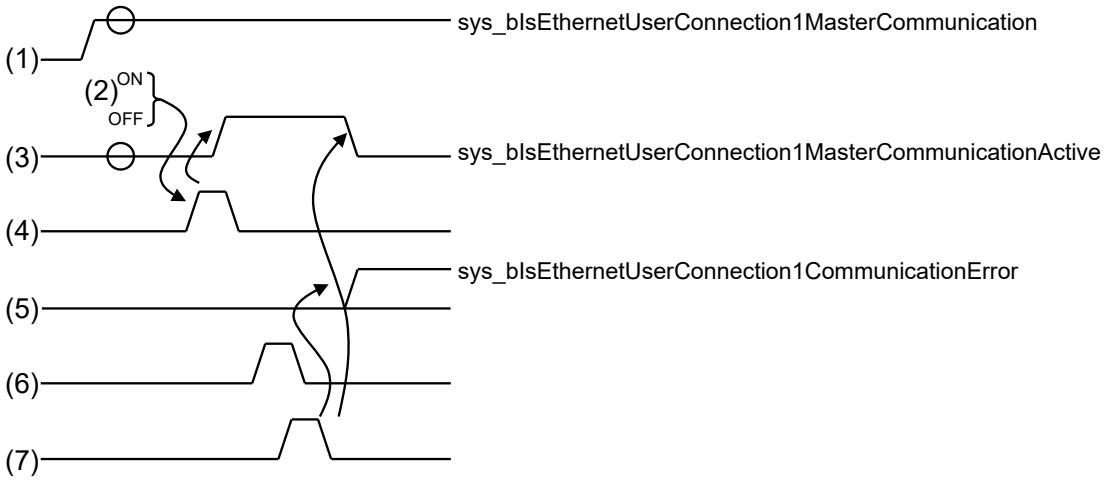
**Destination** (ANY)

Word area or register on the master unit to which the requested data is written.

**Result** (ANY)

For FP7 only: For a description of all error codes, please refer to the table of Modbus/ MEWTOCOL communication error codes.
For other PLCs: set to 0

## Time chart




(1) Master communication clear-to-send flag,e.g.
**sys_bIsEthernetUserConnection1MasterCommunication**

(2) Check that the master communication clear-to-send flag is TRUE and check that the master communication sending flag is FALSE

(3) Master communication sending flag, e.g.
**sys_bIsEthernetUserConnection1MasterCommunicationActive**
While sending: Master communication sending flag is TRUE
Sending done: Master communication sending flag is FALSE

(4) Execute this instruction

(5) Master communication sending done flag, e.g.
**sys_bIsEthernetUserConnection1CommunicationError**
Normal completion: FALSE
Abnormal completion: TRUE

(6) Send data

(7) Processing of the received response

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- If slave or master data exceed the available address range.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- If slave or master data exceed the available address range.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global variables*

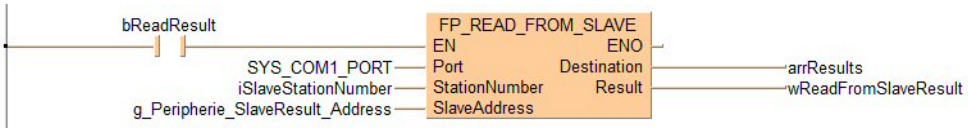
| | Class | Identifier | FP address | IEC address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | g_Peripherie_SlaveResult_Address | DDT200 | %MD5.200 | ARRAY [0..9] OF REAL | [10(0)] |

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_Peripherie_SlaveResult_Address | ARRAY [0..9] OF REAL | [10(0)] | |
| 1 | VAR | bReadResult | BOOL | FALSE | |
| 2 | VAR | wReadFromSlaveResult | WORD | 0 | result of write to slave instruction |
| 3 | VAR | iSlaveStationNumber | INT | 0 | slave station number |
| 4 | VAR | arrResults | ARRAY [0..9] OF REAL | [10(0.0)] | Array of results to be read from slave station |
| 5 | VAR | iSlaveMemoryArea | INT | 5 | memory Area in slave station |
| 6 | VAR | iSlaveMemoryOffset | INT | 100 | |
| 7 | VAR | iSlaveMemorySize | INT | 1 | |

### LD body

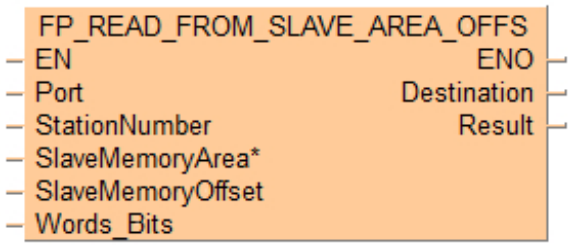


# FP_READ_FROM_SLAVE_AREA_OFFS

Read data from slave with offset

Use this instruction to request data from a slave via the communication port using the MEWTOCOL-COM or Modbus RTU protocol, as defined in the system registers of the port used. Make sure the same protocol is set for master and slave. Master and slave must have matching memory areas.

For data transmissions using the Modbus protocol, the compiler generates Modbus commands based on the Modbus reference numbers.




## Parameters

Input

**Port** (INT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**StationNumber** (WORD, INT, UINT)

Station number of the slave (MEWTOCOL: 1–99, MODBUS: 1–247)
Set to 1, if a **SYS_ETHERNET_USER_CONNECTION_xx** is applied to input **Port**

**SlaveMemoryArea** (WORD, INT, UINT)

Starting address on the slave from which the data is requested.

| Memory area type | System variable |
|---|---|
| Internal flags | SYS_MEMORY_AREA_R |
| Link flags | SYS_MEMORY_AREA_L |
| External inputs | SYS_MEMORY_AREA_X |
| External outputs | SYS_MEMORY_AREA_Y |
| Internal flags | SYS_MEMORY_AREA_WR |

| Memory area type | System variable |
|---|---|
| Timer/counter set value | SYS_MEMORY_AREA_SV |
| Timer/counter elapsed value | SYS_MEMORY_AREA_EV |
| Data registers | SYS_MEMORY_AREA_DT |
| Link flags | SYS_MEMORY_AREA_WL |
| Link registers | SYS_MEMORY_AREA_LD |
| File registers | SYS_MEMORY_AREA_FL |
| Input registers | SYS_MEMORY_AREA_WX |
| Output registers | SYS_MEMORY_AREA_WY |

**SlaveMemoryOffset** (WORD, INT, UINT)

Offset of the starting address on the slave from which the data is requested.

**Words_Bits** (WORD, INT, UINT)

Number of words (bits) to be transmitted.
Either:

- Number of words
  for Modbus RTU: 16#001–16#07F
  for MEWTOCOL-COM: 16#001–16#1FD or 16#001–16#1B (FP0, FP-e)

Or:

- Control word for bit transfer: 16#8T0F with **T** for a bit transfer to the master and **F** for a bit transfer to the slave (does not apply to FP7).

Output

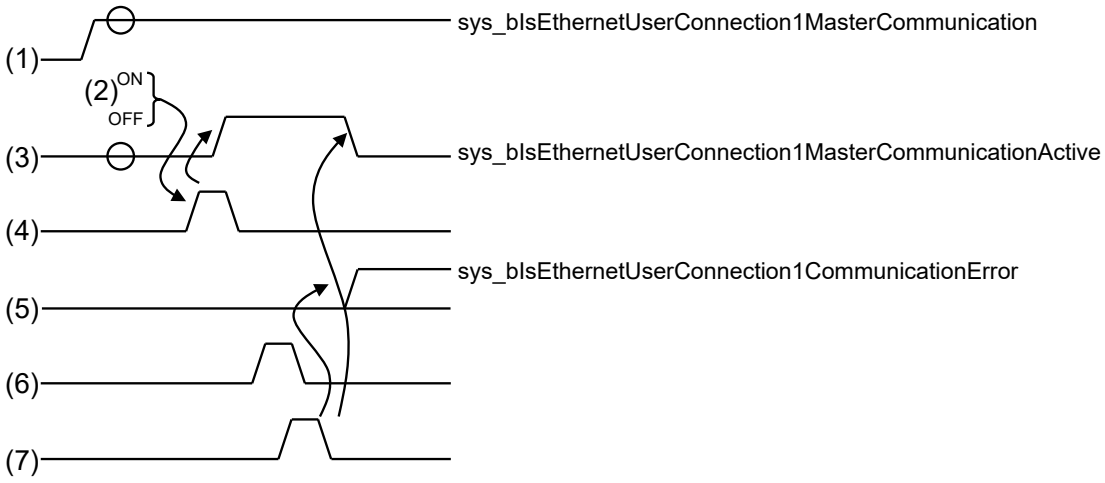
**Destination**(ANY)

Word area or register on the master unit to which the requested data is written.

**Result**(ANY)

For FP7 only: For a description of all error codes, please refer to the table of Modbus/MEWTOCOL communication error codes.
For other PLCs: set to 0

## Time chart




(1) Master communication clear-to-send flag,e.g. **sys_bIsEthernetUserConnection1MasterCommunication**

(2) Check that the master communication clear-to-send flag is TRUE and check that the master communication sending flag is FALSE

(3) Master communication sending flag, e.g. **sys_bIsEthernetUserConnection1MasterCommunicationActive**
While sending: Master communication sending flag is TRUE
Sending done: Master communication sending flag is FALSE

(4) Execute this instruction

(5) Master communication sending done flag, e.g. **sys_bIsEthernetUserConnection1CommunicationError**
Normal completion: FALSE
Abnormal completion: TRUE

(6) Send data

(7) Processing of the received response

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- If slave or master data exceed the available address range.
- if the number of sent data specified by **Words_Bits** is incorrect.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- If slave or master data exceed the available address range.
- if the number of sent data specified by **Words_Bits** is incorrect.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.

- If the COM port selected requires a communication cassette that has not been installed.

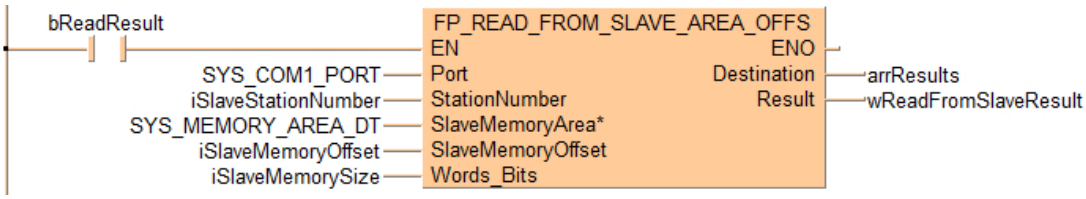
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bReadResult | BOOL | FALSE | |
| 1 | VAR | wReadFromSlaveResult | WORD | 0 | result of write to slave instruction |
| 2 | VAR | iSlaveStationNumber | INT | 0 | slave station number |
| 3 | VAR | arrResults | ARRAY [0..9] OF REAL | [10(0.0)] | Array of results to be read from slave station |
| 4 | VAR | iSlaveMemoryOffset | INT | 200 | |
| 5 | VAR | iSlaveMemorySize | INT | 1 | |
| 6 | VAR_CONSTANT | iSlaveMemoryArea | INT | 5 | memory Area in slave station |

### LD body

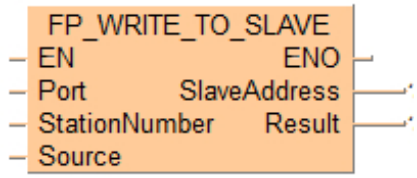


# FP_WRITE_TO_SLAVE

Write data to slave

Use this instruction to write data from a master to a slave via the communication port using the MEWTOCOL-COM or Modbus RTU protocol, as defined in the system registers of the port used. Make sure the same protocol is set for master and slave. Master and slave must have matching memory areas. If the slave data is not available in the user area of the master, use **FP_WRITE_TO_SLAVE_AREA_OFFS** or **FP_MODBUS_MASTER**.

For data transmissions using the Modbus protocol, the compiler generates Modbus commands based on the Modbus reference numbers.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**StationNumber** (WORD, INT, UINT)

Station number of the slave (MEWTOCOL: 1-99, MODBUS: 1-247)
Set to 1, if a **SYS_ETHERNET_USER_CONNECTION_xx** is applied to input **Port**

**Source** (WORD, INT, UINT)

Word area or register on the master unit for the data to be written to the slave.

Output

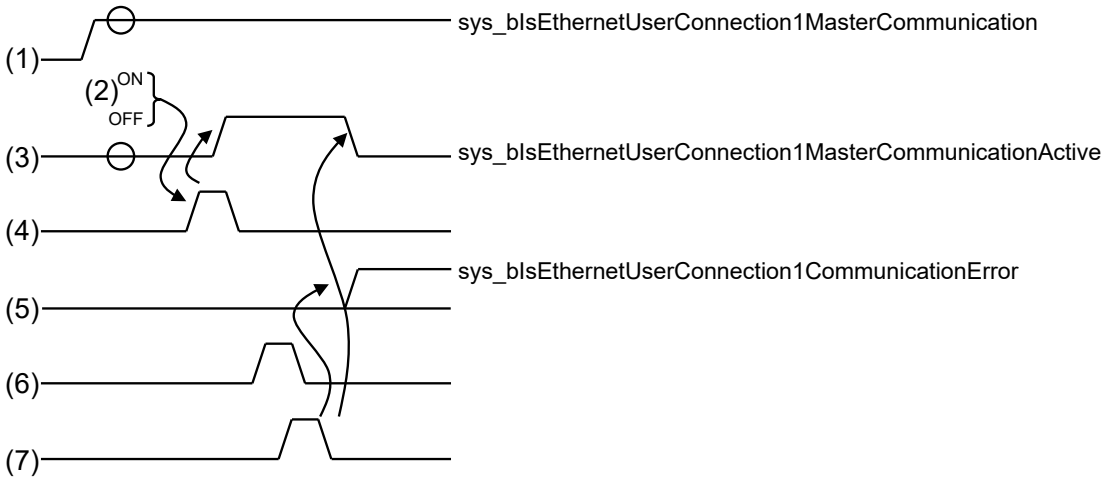
**SlaveAddress**(ANY)

Destination address on the slave to which the requested data is written.

**Result**(ANY)

For FP7 only: For a description of all error codes, please refer to the table of Modbus/ MEWTOCOL communication error codes.
For other PLCs: set to 0

## Time chart




(1) Master communication clear-to-send flag,e.g. **sys_bIsEthernetUserConnection1MasterCommunication**

(2) Check that the master communication clear-to-send flag is TRUE and check that the master communication sending flag is FALSE

(3) Master communication sending flag, e.g. **sys_bIsEthernetUserConnection1MasterCommunicationActive**
While sending: Master communication sending flag is TRUE
Sending done: Master communication sending flag is FALSE

(4) Execute this instruction

(5) Master communication sending done flag, e.g. **sys_bIsEthernetUserConnection1CommunicationError**
Normal completion: FALSE
Abnormal completion: TRUE

(6) Send data

(7) Processing of the received response

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- If slave or master data exceed the available address range.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- If slave or master data exceed the available address range.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global variables*

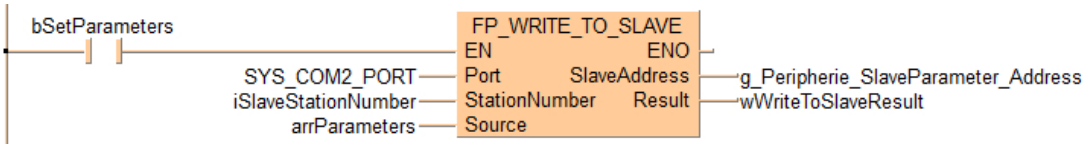
| | Class | Identifier | FP address | IEC address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | g_Peripherie_SlaveParameter_Address | DDT100 | %MD5.100 | ARRAY [0..15] OF DINT | [16(0)] |
| 1 | VAR_GLOBAL | g_Peripherie_SlaveResult_Address | DDT200 | %MD5.200 | ARRAY [0..9] OF REAL | [10(0)] |

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_Peripherie_SlaveResult_Address | ARRAY [0..9] OF REAL | [10(0)] | |
| 1 | VAR_EXTERNAL | g_Peripherie_SlaveParameter_Address | ARRAY [0..15] OF D... | [16(0)] | |
| 2 | VAR | wWriteToSlaveResult | WORD | 0 | result of write to slave instruction |
| 3 | VAR | iSlaveStationNumber | INT | 0 | slave station number |
| 4 | VAR | arrParameters | ARRAY [0..15] OF D... | | Array of parameters to be send to slave station |
| 5 | VAR | bSetParameters | BOOL | FALSE | |

### LD body

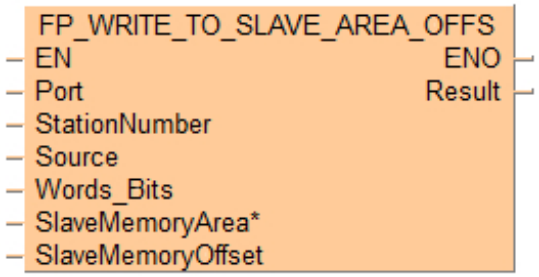


# FP_WRITE_TO_SLAVE_AREA_OFFS

Write data to slave with offset

Use this instruction to write data from a master to a slave via the communication port using the MEWTOCOL-COM or Modbus RTU protocol, as defined in the system registers of the port used. Make sure the same protocol is set for master and slave. Master and slave must have matching memory areas.

For data transmissions using the Modbus protocol, the compiler generates Modbus commands based on the Modbus reference numbers.




## Parameters

Input

**Port** (INT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**StationNumber** (WORD, INT, UINT)

Station number of the slave (MEWTOCOL: 1–99,MODBUS: 1–247)
Set to 1, if a **SYS_ETHERNET_USER_CONNECTION_xx** is applied to input **Port**

**Source** (WORD, INT, UINT)

Word area or register on the master unit for the data to be written to the slave.

**Words_Bits** (WORD, INT, UINT)

Number of words (bits) to be transmitted.
Either:

- Number of words
  for Modbus RTU: 16#001–16#07F
  for MEWTOCOL-COM: 16#001–16#1FD or 16#001–16#1B (FP0, FP-e)

Or:

- Control word for bit transfer: 16#8T0F with **T** for a bit transfer to the master and **F** for a bit transfer to the slave (does not apply to FP7).

**SlaveMemoryArea** (WORD, INT, UINT)

Memory area on the slave where to store the data (destination = **SlaveMemoryArea** + **SlaveMemoryOffset**).

| Memory area type | System variable |
|---|---|
| Internal flags | **SYS_MEMORY_AREA_R** |
| Link flags | **SYS_MEMORY_AREA_L** |
| External inputs | **SYS_MEMORY_AREA_X** |
| External outputs | **SYS_MEMORY_AREA_Y** |
| Internal flags | **SYS_MEMORY_AREA_WR** |
| Timer/counter set value | **SYS_MEMORY_AREA_SV** |
| Timer/counter elapsed value | **SYS_MEMORY_AREA_EV** |
| Data registers | **SYS_MEMORY_AREA_DT** |
| Link flags | **SYS_MEMORY_AREA_WL** |
| Link registers | **SYS_MEMORY_AREA_LD** |
| File registers | **SYS_MEMORY_AREA_FL** |
| Input registers | **SYS_MEMORY_AREA_WX** |
| Output registers | **SYS_MEMORY_AREA_WY** |

**SlaveMemoryOffset** (WORD, INT, UINT)

Offset of the memory area where to store the data in the slave (destination = **SlaveMemoryArea** + **SlaveMemoryOffset**).

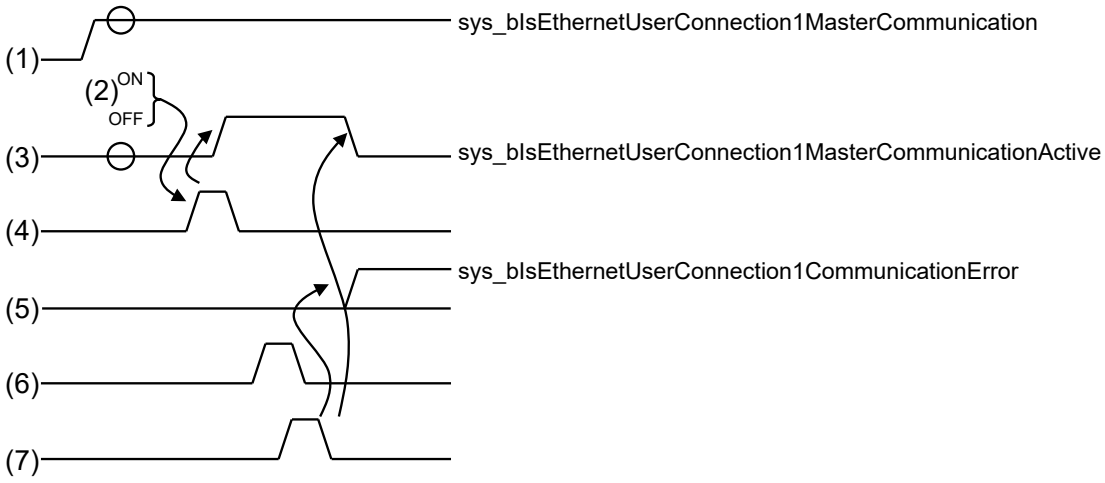
Output

**Result** (WORD, INT, UINT)

For FP7 only: For a description of all error codes, please refer to the table of Modbus/ MEWTOCOL communication error codes.

For other PLCs: set to 0

## Time chart




(1) Master communication clear-to-send flag,e.g. **sys_bIsEthernetUserConnection1MasterCommunication**

(2) Check that the master communication clear-to-send flag is TRUE and check that the master communication sending flag is FALSE

(3) Master communication sending flag, e.g. **sys_bIsEthernetUserConnection1MasterCommunicationActive**
While sending: Master communication sending flag is TRUE
Sending done: Master communication sending flag is FALSE

(4) Execute this instruction

(5) Master communication sending done flag, e.g. **sys_bIsEthernetUserConnection1CommunicationError**
Normal completion: FALSE
Abnormal completion: TRUE

(6) Send data

(7) Processing of the received response

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- If slave or master data exceed the available address range.
- if the number of sent data specified by **Words_Bits** is incorrect.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- If slave or master data exceed the available address range.
- if the number of sent data specified by **Words_Bits** is incorrect.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.

- If the COM port selected requires a communication cassette that has not been installed.

## Example

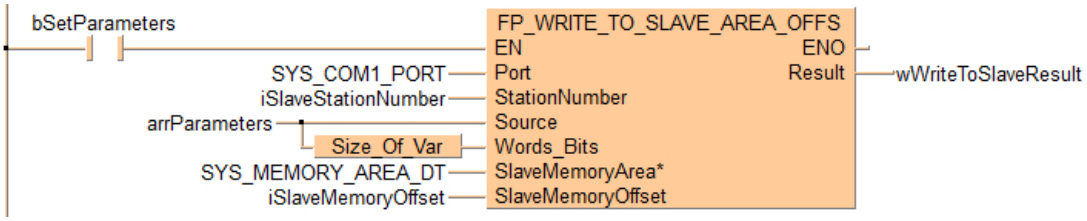
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bSetParameters | BOOL | FALSE | |
| 1 | VAR | wWriteToSlaveResult | WORD | 0 | result of write to slave instruction |
| 2 | VAR | iSlaveStationNumber | INT | 0 | slave station number |
| 3 | VAR | arrParameters | ARRAY [0..15] OF DINT | | Array of parameters to be send to slave station |
| 4 | VAR | iSlaveMemoryOffset | INT | 100 | |
| 5 | VAR | bBool | BOOL | FALSE | |
| 6 | VAR_CONSTANT | iSlaveMemoryArea_Bool | INT | 10000 | memory Area in slave station |
| 7 | VAR_CONSTANT | iSlaveMemoryArea_Word | INT | 5 | memory Area in slave station |

### POU body

### LD body



## 7.6 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

Advantages of IEC instructions

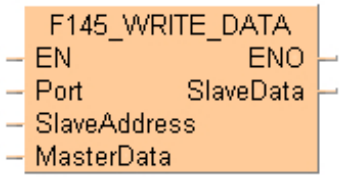
Advantages of FP instructions

# F145_WRITE_DATA

Write data to slave

Use this instruction to write data from a master to a slave via the communication port using the MEWTOCOL-COM or Modbus RTU protocol, as defined in the system registers of the port used. Make sure the same protocol is set for master and slave. Master and slave must have matching memory areas. If the slave data is not available in the user area of the master, use **F145_WRITE_DATA_TYPE_OFFS** or **F145F146_MODBUS_MASTER**.

For data transmissions using the Modbus protocol, the compiler generates Modbus commands based on the Modbus reference numbers.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**SlaveAddress** (WORD, INT, UINT)

Station number of the slave (MEWTOCOL: 0–99, MODBUS: 0–255, 0: Broadcasting)
Set to 1, if a **SYS_ETHERNET_USER_CONNECTION_xx** is applied to input **Port**

**MasterData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Source address on the master for the data to be written to the slave.

Output

**SlaveData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

- Destination address on the slave to which the data is written.
- **MasterData** and **SlaveData** must be of the same data type.
- To establish external data access from the master to the slave data you must assign fixed user addresses (same addresses as slave data) in the global variable list.

### Remarks

- Instead of using this F instruction, we recommend using the most flexible instruction:**FP_WRITE_TO_SLAVE_AREA_OFFS**.
- The F145 or F146 instructions can only be executed if neither instruction is active. Evaluate the "F145/F146 not active" flag in your program to check the state of the instructions.
- The F145 instruction only requests that data be sent to the slave. The actual processing takes place at the end of the scan.
- Evaluate the "F145/F146 error" flag to check whether transmission has completed normally or with an error.
- When broadcasting (**SlaveAddress** set to 0), make sure transmission is executed only after the maximum scan time has elapsed.
- The F145 or F146 instructions cannot be executed if the destination address is a special internal flag (from R9000), a special data register (from DT9000/DT90000), or a file register FL.

### Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- If slave or master data exceed the available address range.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- If slave or master data exceed the available address range.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

### Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

| | Class | Identifier | FP Address | IEC Address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | Slave2_g_bY38 | Y38 | %QX3.8 | BOOL | FALSE |

### POU header

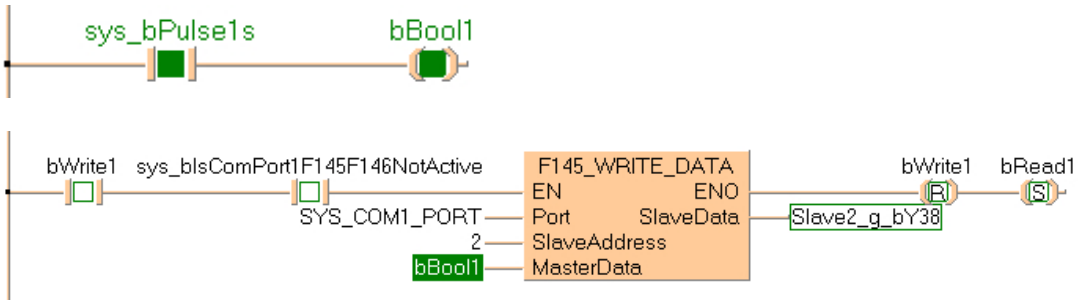
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bBool1 | BOOL | FALSE |
| 1 | VAR | bRead1 | BOOL | FALSE |
| 2 | VAR | bWrite1 | BOOL | FALSE |
| 3 | VAR_EXTERNAL | Slave2_g_bY38 | BOOL | FALSE |

### POU body

The system variable **sys_bPulse1s** is copied to **bBool1**. If **bWrite1** and **sys_bIsComPort1F145F146NotActive** are set to TRUE, **bBool1** is written to the output Y38 of slave 2.

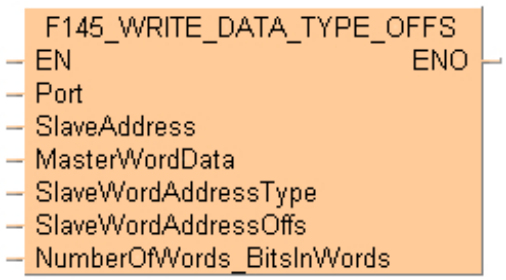
### LD body



# F145_WRITE_DATA_TYPE_OFFS

Write data to slave with type and offset

Use this instruction to write data from a master to a slave via the communication port using the MEWTOCOL-COM or Modbus RTU protocol, as defined in the system registers of the port used. For data transmissions using the Modbus protocol, the compiler generates Modbus commands based on the Modbus reference numbers.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**SlaveAddress** (WORD, INT, UINT)

Station number of the slave (MEWTOCOL: 1–99, MODBUS: 1–255, Broadcasting: 0)
Set to 1, if a **SYS_ETHERNET_USER_CONNECTION_xx** is applied to input **Port**

**MasterWordData** (ANY)

Source address on the master for the data to be written to the slave.

**SlaveWordAddressType** (WORD, INT, UINT)

Destination address type in the slave. Specify an offset of zero, e.g. DT0 or WL0.

**SlaveWordAddressOffs** (WORD, INT, UINT)

Offset of the starting address on the slave to which the data is written. The address type is defined by **SlaveWordAddressType**.

**NumberOfWords_BitsInWords** (WORD, INT, UINT)

Number of words (bits) to be transmitted.

Either:

- Number of words

  for Modbus RTU: 16#001–16#07F

  for MEWTOCOL-COM: 16#001–16#1FD or 16#001–16#1B (FP0, FP-e)

Or:

- Control word for bit transfer: 16#8T0F with **T** for a bit transfer to the master and **F** for a bit transfer to the slave (does not apply to FP7).

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_WRITE_TO_SLAVE_AREA_OFFS**.
- The F145 or F146 instructions can only be executed if neither instruction is active. Evaluate the "F145/F146 not active" flag in your program to check the state of the instructions.
- The F145 instruction only requests that data be sent to the slave. The actual processing takes place at the end of the scan.
- Evaluate the "F145/F146 error" flag to check whether transmission has completed normally or with an error.
- When broadcasting (**SlaveAddress** set to 0), make sure transmission is executed only after the maximum scan time has elapsed.
- The F145 or F146 instructions cannot be executed if the destination address is a special internal flag (from R9000), a special data register (from DT9000/DT90000), or a file register FL.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- If slave or master data exceed the available address range.
- If **SlaveWordAddressType**: Offset ≠ 0
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- If slave or master data exceed the available address range.
- If **SlaveWordAddressType**: Offset ≠ 0
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.

- If the COM port selected requires a communication cassette that has not been installed.

## Example

### POU header

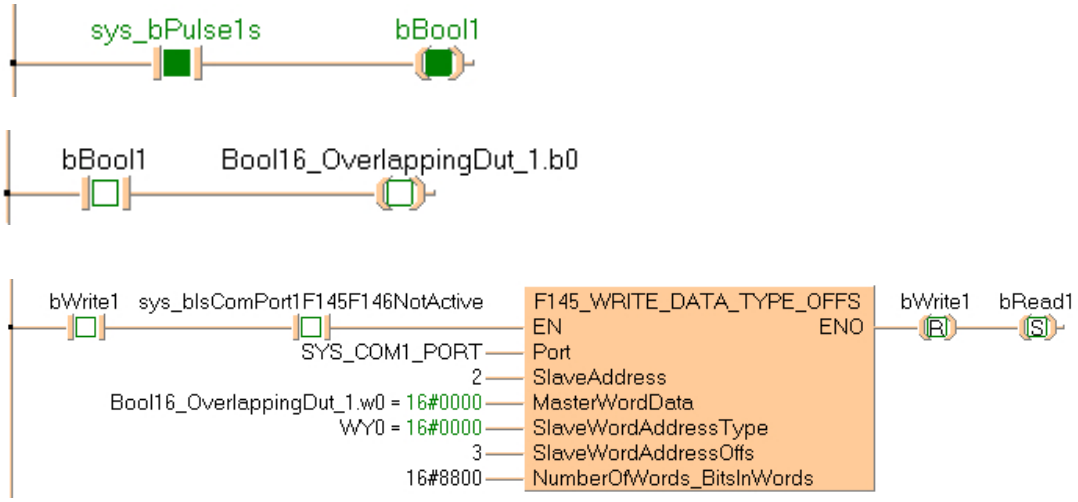
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bWrite1 | BOOL | TRUE |
| 1 | VAR | bRead1 | BOOL | FALSE |
| 2 | VAR | bBool1 | BOOL | FALSE |
| 3 | VAR | Bool16_OverlappingDut_1 | BOOL16_OVERLAPPING_DUT | |

### POU body

The system variable **sys_bPulse1s** is copied to **bBool1** and **Bool16_OverlappingDut_1.b0**. If **bWrite1** and **sys_bIsComPort1F145F146NotActive** are set to TRUE, **bBool1** is written to the output Y38 of slave 2 via **Bool16_OverlappingDut_1.b0**.

### LD body

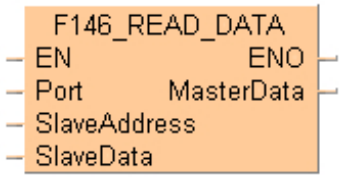


# F146_READ_DATA

Read data from slave

Use this instruction to request data from a slave via the communication port using the MEWTOCOL-COM or Modbus RTU protocol, as defined in the system registers of the port used.Make sure the same protocol is set for master and slave.Master and slave must have matching memory areas. If the slave data is not available in the user area of the master, use **F146_READ_DATA_TYPE_OFFS** or **F145F146_MODBUS_MASTER**.

For data transmissions using the Modbus protocol, the compiler generates Modbus commands based on the Modbus reference numbers.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**SlaveAddress** (WORD, INT, UINT)

Station number of the slave (MEWTOCOL: 1–99, MODBUS: 1–255)
Set to 1, if a **SYS_ETHERNET_USER_CONNECTION_xx** is applied to input **Port**

**SlaveData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Source address on the slave from which the data is requested.

Output

**MasterData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Word area or register on the master unit to which the requested data is written.

### Remarks

- Instead of using this F instruction, we recommend using the most flexible instruction:**FP_READ_FROM_SLAVE_AREA_OFFS**.
- The F145 or F146 instructions can only be executed if neither instruction is active. Evaluate the "F145/F146 not active" flag in your program to check the state of the instructions.
- Evaluate the "F145/F146 error" flag to check whether transmission has completed normally or with an error.
- The F145 or F146 instructions cannot be executed if the destination address is a special internal flag (from R9000), a special data register (from DT9000/DT90000), or a file register FL.

### Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- If slave or master data exceed the available address range.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- If slave or master data exceed the available address range.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

### Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

| | Class | Identifier | FP Address | IEC Address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | Slave2_g_bY38 | Y38 | %QX3.8 | BOOL | FALSE |

### POU header

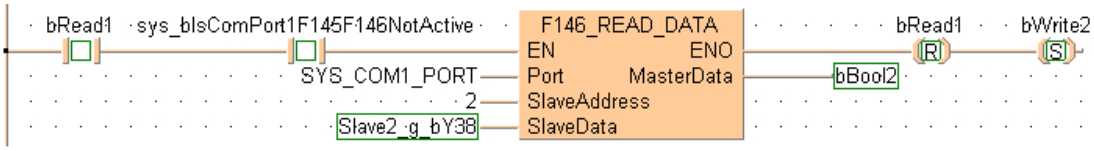
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | Slave2_g_bY38 | BOOL | FALSE |
| 1 | VAR | bRead1 | BOOL | FALSE |
| 2 | VAR | bWrite2 | BOOL | FALSE |
| 3 | VAR | bBool2 | BOOL | FALSE |

## POU body

If **bRead1** and **sys_bIsComPort1F145F146NotActive** are set to TRUE, the global variable **Slave2_g_bY38**, which is assigned to Y38 of slave 2, is read and stored in **bBool2**.

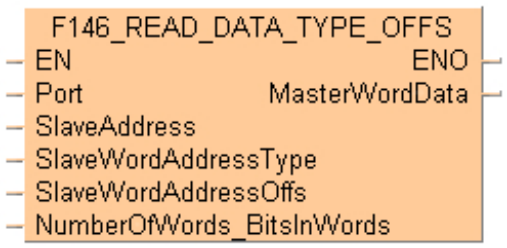
## LD body



# F146_READ_DATA_TYPE_OFFS

Read data from slave with type and offset

Use this instruction to request data from a slave via the communication port using the MEWTOCOL-COM or Modbus RTU protocol, as defined in the system registers of the port used. For data transmissions using the Modbus protocol, the compiler generates Modbus commands based on the Modbus reference numbers.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**SlaveAddress** (WORD, INT, UINT)

Station number of the slave (MEWTOCOL: 1–99, MODBUS: 1–255), Broadcasting: 0)
Set to 1, if a **SYS_ETHERNET_USER_CONNECTION_xx** is applied to input **Port**

**SlaveWordAddressType** (WORD, INT, UINT)

Destination address type in the slave. Specify an offset of zero, e.g. DT0 or WL0.

**SlaveWordAddressOffs** (WORD, INT, UINT)

Offset of the starting address on the slave from which the data is read. The address type is defined by **SlaveWordAddressType**.

**NumberOfWords_BitsInWords** (WORD, INT, UINT)

Number of words (bits) to be transmitted.
Either:

- Number of words
  for Modbus RTU: 16#001–16#07F

for MEWTOCOL-COM: 16#001–16#1FD or 16#001–16#1B (FP0, FP-e)

Or:

- Control word for bit transfer: 16#8T0F with **T** for a bit transfer to the master and **F** for a bit transfer to the slave (does not apply to FP7).

Output

**MasterWordData** (ANY)

Word area or register on the master unit to which the requested data is written.

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_READ_FROM_SLAVE_AREA_OFFS**.
- The F145 or F146 instructions can only be executed if neither instruction is active. Evaluate the "F145/F146 not active" flag in your program to check the state of the instructions.
- Evaluate the "F145/F146 error" flag to check whether transmission has completed normally or with an error.
- The F145 or F146 instructions cannot be executed if the destination address is a special internal flag (from R9000), a special data register (from DT9000/DT90000), or a file register FL.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- If slave or master data exceed the available address range.
- If **SlaveWordAddressType**: Offset ≠ 0
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- If slave or master data exceed the available address range.
- If **SlaveWordAddressType**: Offset ≠ 0
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

## Example

### POU header

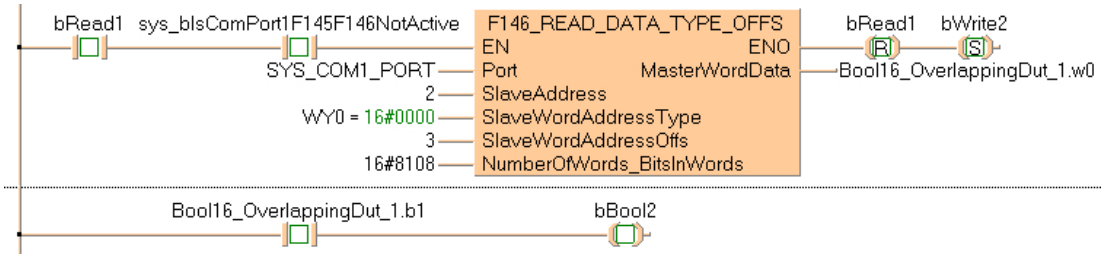
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bRead1 | BOOL | FALSE |
| 1 | VAR | bWrite2 | BOOL | FALSE |
| 2 | VAR | bBool2 | BOOL | FALSE |
| 3 | VAR | Bool16_OverlappingDut_1 | BOOL16_OVERLAPPING_DUT | |

### POU body

If **bRead1** and **sys_bIsComPort1F145F146NotActive** are set to TRUE, the output Y38 of slave 2 is read and written to bit 1 of **Bool16_OverlappingDut_1.w0**. This bit can be accessed by **Bool16_OverlappingDut_1.b1** and is copied to **bBool2**.

### LD body



# F145F146_MODBUS_MASTER

Write data to slave or read data from slave

Available function codes (with programming examples)

**SYS_MODBUS_01_READ_COIL** (page 416)

**SYS_MODBUS_02_READ_INPUT** (page 416)

**SYS_MODBUS_03_READ_HOLDING_REGISTER** (page 417)

**SYS_MODBUS_04_READ_INPUT_REGISTERS** (page 418)

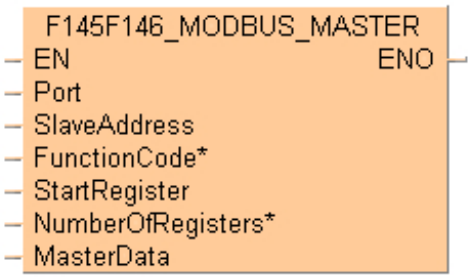
**SYS_MODBUS_05_FORCE_COIL** (page 418)

**SYS_MODBUS_06_PRESET_REGISTER** (page 419)

**SYS_MODBUS_15_FORCE_COILS** (page 419)

**SYS_MODBUS_16_PRESET_REGISTERS** (page 420)

Use this instruction to write data from a master to a slave or read data from a slave via the communication port using the Modbus RTU protocol, as defined in the system registers of the port used. Make sure the same protocol is set for master and slave.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**SlaveAddress** (WORD, INT, UINT)

Station number of the slave (0–255).
Set to 1, if a **SYS_ETHERNET_USER_CONNECTION_xx** is applied to input **Port**

**FunctionCode*** (WORD, INT, UINT)

**SYS_MODBUS_01_READ_COIL**
**SYS_MODBUS_02_READ_INPUT**
**SYS_MODBUS_03_READ_HOLDING_REGISTER**
**SYS_MODBUS_04_READ_INPUT_REGISTERS**
**SYS_MODBUS_05_FORCE_COIL**
**SYS_MODBUS_06_PRESET_REGISTER**
**SYS_MODBUS_15_FORCE_COILS**
**SYS_MODBUS_16_PRESET_REGISTERS**

**StartRegister** (WORD, INT, UINT)

Starting address (0–65535). The address type depends on the command specified by **FunctionCode***.

**NumberOfRegisters*** (WORD, INT, UINT)

Number of transmission bits or words.

- 1–2040 for function codes 01, 02
- 2–2040 for function code 15
- 1–127 for function codes 03, 04
- 2–127 for function code 16

**MasterData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Source address on the master for the data to be written to the slave.

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_MODBUS_MASTER**
- In contrast to other F145 or F146 instructions, the required Modbus command can directly be set by the parameter **FunctionCode***.
- The F145 or F146 instructions can only be executed if neither instruction is active. Evaluate the "F145/F146 not active" flag in your program to check the state of the instructions.
- The F145 instruction only requests that data be sent to the slave. The actual processing takes place at the end of the scan.
- Evaluate the "F145/F146 error" flag to check whether transmission has completed normally or with an error.
- When broadcasting (**SlaveAddress** set to 0), make sure transmission is executed only after the maximum scan time has elapsed.
- The F145 or F146 instructions cannot be executed if the destination address is a special internal flag (from R9000), a special data register (from DT9000/DT90000), or a file register FL.

- Commands supported by the master:

| Function code | System constant | Start register | Number of registers | Reference numbers (depending on Modbus slave) |
|---|---|---|---|---|
| 01 | **SYS_MODBUS_01_ READ_COIL** | 0–65535 | 1–2040 | 000001–065536 |
| 02 | **SYS_MODBUS_02_ READ_INPUT** | 0–65535 | 1–2040 | 100001–165536 |
| 03 | **SYS_MODBUS_03_ READ_HOLDING_ REGISTER** | 0–65535 | 1–127 | 400001–465536 |
| 04 | **SYS_MODBUS_04_ READ_INPUT_ REGISTERS** | 0–65535 | 1–127 | 300001–365536 |
| 5 | **SYS_MODBUS_05_ FORCE_COIL** | 0–65535 | 1 | 000001–065536 |
| 6 | **SYS_MODBUS_06_ PRESET_REGISTER** | 0–65535 | 1 | 400001–465536 |
| 15 | **SYS_MODBUS_15_ FORCE_COILS** | 0–65535 | 2–2040 | 000001–065536 |
| 16 | **SYS_MODBUS_16_ PRESET_REGISTERS** | 0–65535 | 2–127 | 400001–465536 |

- Modbus specifications for Panasonic PLCs:

| Modbus area address | Modbus memory name | Memory type | Panasonic PLCs address |
|---|---|---|---|
| 000001 … | COIL | 1-bit | Y0 … |
| 002049 … | | | R0 … |
| 100001 … | INPUT | 1-bit | X0 … |
| 400001 … | HOLDING_REGISTER | 16-bit | DT0 … |
| 300001 … | INPUT_REGISTERS | 16-bit | WL0 … |
| 302001 … | | | LD0 … |

Tip

For reference number and address area ranges supported by the Panasonic PLCs, please refer to the User's Manual of the PLC. If the reference number is outside the supported range, an error is returned.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- If slave or master data exceed the available address range.

- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- If slave or master data exceed the available address range.
- If the communication mode is **not** set to MEWTOCOL-COM Master/Slave or Modbus RTU Master/Slave.
- If the COM port selected requires a communication cassette that has not been installed.

## Example

### SYS_MODBUS_01_READ_COIL

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

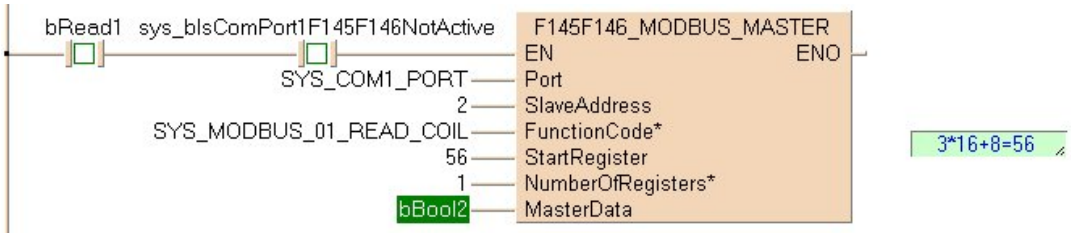
For the special case that the connected slave is a Panasonic PLC configured as Modbus RTU slave via system register, one or multiple bits are read from:

- Y (Output)
- R (Internal flags)

#### Example

Executing Modbus command 01: reads 1 bit from a Modbus slave beginning at start register 56 set by the variable **StartRegister**. Then the command stores the 1 bit in the master beginning at the address set by the variable **bBool2**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, the start register 56 corresponds to output **Y38** (3*16+8=56).




### SYS_MODBUS_02_READ_INPUT

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

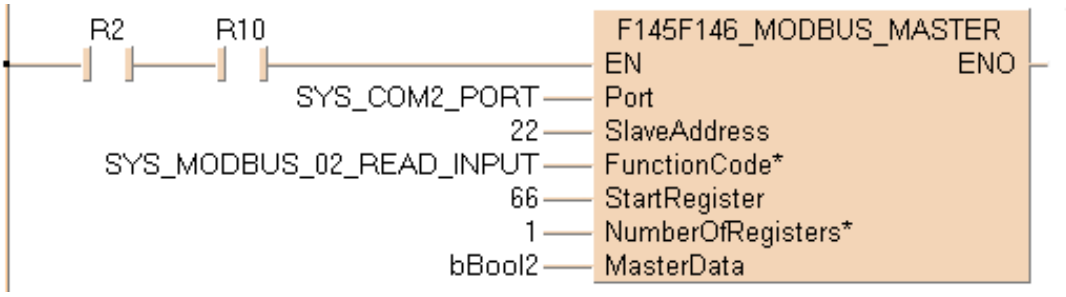
For the special case that the connected slave is a Panasonic PLC configured as Modbus RTU slave via system register, one or multiple bits are read from:

- X (Input)

### Example

Executing Modbus command 02: reads 1 bit from a Modbus slave beginning at start register 66 set by the variable **StartRegister**. Then the command stores the 1 bit in the master beginning at the address set with the variable **bBool2**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, start register 66 corresponds to input **X42** (4*16+2=66).




## SYS_MODBUS_03_READ_HOLDING_REGISTER

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

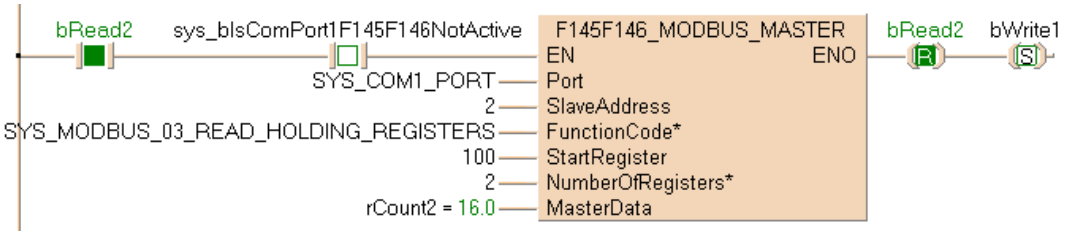
For the special case that the connected slave is a Panasonic PLC configured as Modbus RTU slave via system register, one or multiple bits are read from:

- DT (data registers)

### Example

Executing Modbus command 03: reads 2 words from a Modbus slave beginning at start register 100 set by the variable **StartRegister**. Then the command stores the 2 words in the Modbus master 2 beginning at the address set by the variable **rCount2**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, start register 100 corresponds to data register DDT100.




Maximum number of registers: 127.

### SYS_MODBUS_04_READ_INPUT_REGISTERS

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

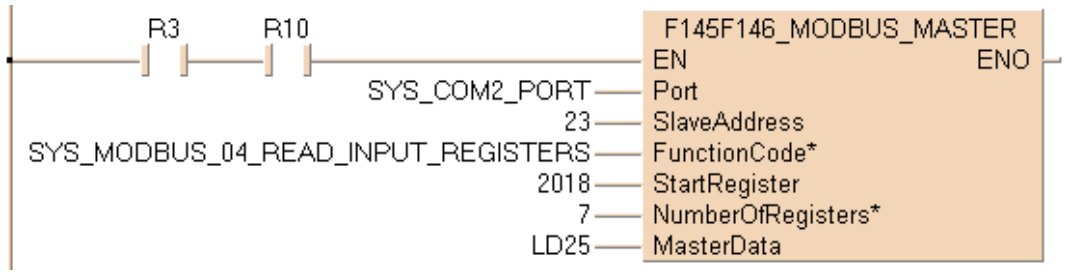
For the special case that the connected slave is a Panasonic PLC configured as Modbus RTU slave via system register, one or multiple bits are read from:

- WL0–WL127 (Link flags)
- LD0–LD256 (Link registers)

#### Example

Executing Modbus command 04: reads 7 words from a Modbus slave beginning at start register 2018 set by the variable **StartRegister**. Then the command stores the 7 words in the master beginning at LD25 set by the variable **MasterData**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, start register 2018 corresponds to link register LD18.




Maximum number of registers: 127.

### SYS_MODBUS_05_FORCE_COIL

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

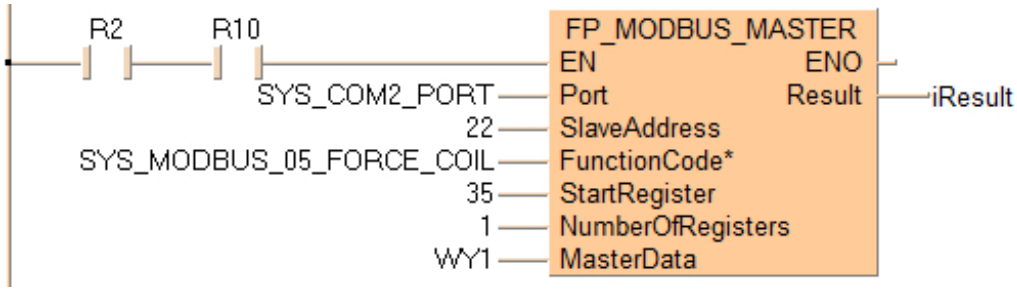
In case the connected slave is a Panasonic PLC in Modbus RTU mode, one or multiple bits are written to:

- Y (Output)
- R (Internal flags)

#### Example

Executing Modbus command 05: writes 1 bit to a Modbus slave beginning at address **WY1** set by the variable **MasterData**. Then the command stores the 1 bit in a Modbus slave beginning at start register 35 set by the variable **StartRegister**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, address 35 corresponds to output **Y23** (2*16+3=35).




When writing multiple bit data use the SYS_MODBUS_15_FORCE_COILS constant.

Maximum number of register: 1

### SYS_MODBUS_06_PRESET_REGISTER

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

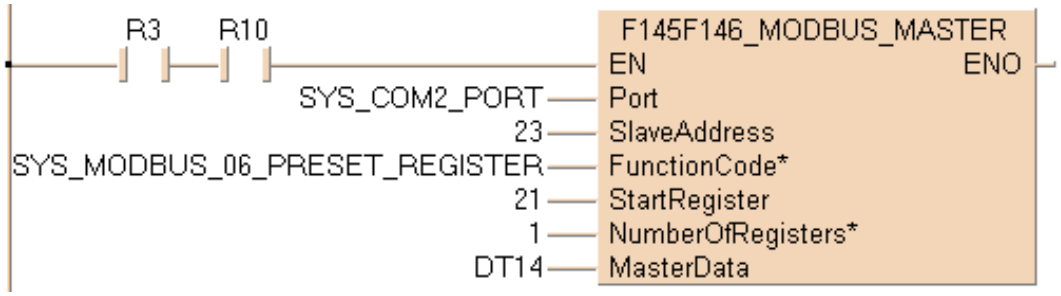
In case the connected slave is a Panasonic PLC in Modbus RTU mode, one or multiple bits are written to:

- DT (data registers)

#### Example

Executing Modbus command 06: writes 1 word to a Modbus slave beginning at address DT14 set by the variable **MasterData**. Then the command stores the 1 word in the Modbus slave beginning at start register 21 set by the variable **StartRegister**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, start register 21 corresponds to data register DT21.




When writing multiple bit data use the SYS_MODBUS_15_FORCE_COILS constant.

Maximum number of register: 1

### SYS_MODBUS_15_FORCE_COILS

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

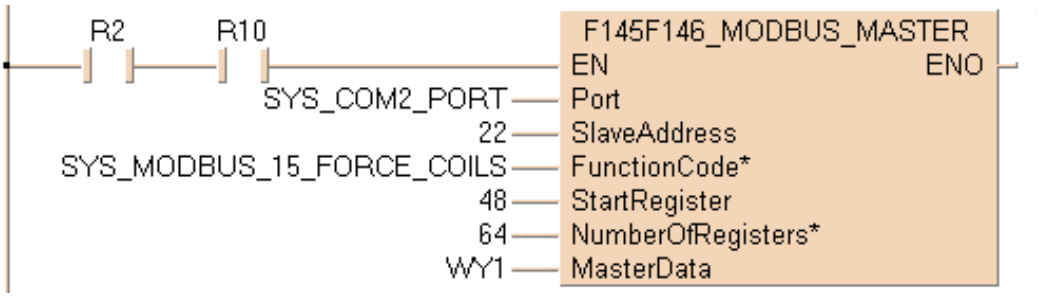
In case the connected slave is a Panasonic PLC in Modbus RTU mode, one or multiple bits are written to:

- Y (Output)
- R (Internal flags)

### Example

Executing Modbus command 15: writes 64 bits to a Modbus slave beginning at address **WY1** set by the variable **MasterData**. Then the command stores the 64 bits in the Modbus slave beginning at start register 48 set by the variable **StartRegister**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, address 48 corresponds to output **Y30** (3*16=48).




## SYS_MODBUS_16_PRESET_REGISTERS

When you apply this constant to the input parameter **FunctionCode***, the corresponding Modbus command is executed.

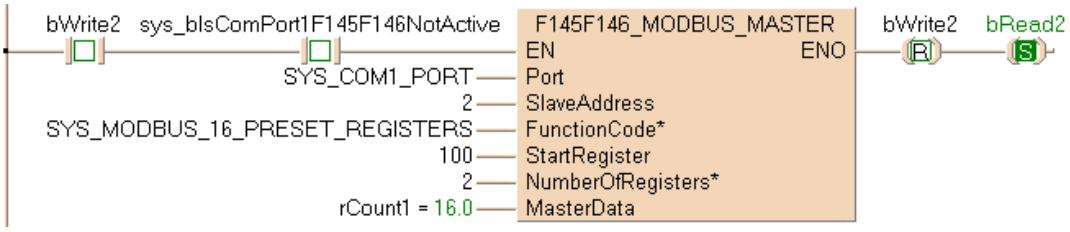
In case the connected slave is a Panasonic PLC in Modbus RTU mode, one or multiple bits are written to:

- DT (data registers)

### Example

Executing Modbus command 16: writes data to a Modbus slave beginning at the address set by the variable **rCount1**. Then the command stores the data in the Modbus slave 2 beginning with start register 100 set by the variable **StartRegister**. The slave's address is converted to a device-specific address depending on the Modbus specifications of the device.

If the connected slave is a Panasonic PLC, start register 100 corresponds to data register DDT100.

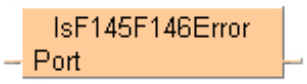



Maximum number of registers: 127.

# IsF145F146Error

Evaluate "F145/F146 error" flag

This instruction returns the value of the "F145/F146 error" flag. It is TRUE if the execution of an F145 or F146 instruction has terminated with an error.




## Parameters

Input

**Port** (WORD, INT, UINT)

Communication port: 1,2

## Remarks

Evaluation of the "F145/F146 error" flag

Evaluate the "F145/F146 error" flag to check whether transmission has completed normally or with an error. The flag can be evaluated using one of the following instructions or system variables:

- **IsF145F146Error**
- **sys_bIsComPort1F145F146Error**
- **sys_bIsComPort2F145F146Error**

For communication ports not supporting the master function, the flag is always FALSE.

Evaluation of the F145/F146 error code

These system variables contain the error code if transmission has terminated with an error.

- **sys_wComPort1F145F146ErrorCode**
- **sys_bIsComPort2F145F146ErrorCode**

For detailed information, please refer to error codes. If the error code is 16#73, a communication timeout error has occurred. The timeout length can be set from 10.0ms–81.9s (in units of 10ms) using system register 32. The default value is 10s.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

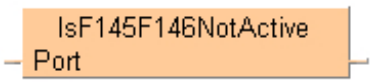
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iPort | INT | 0 |
| 1 | VAR | bF145F146 | BOOL | FALSE |

### LD body

# IsF145F146NotActive

Evaluate "F145/F146 not active" flag

This instruction returns the value of the "F145/F146 not active" flag. It is TRUE if the execution of an F145 or F146 instruction is possible because neither instruction is active.




## Parameters

Input

**Port** (WORD, INT, UINT)

Communication port: 1, 2

## Remarks

Evaluation of the "F145/F146 not active" flag

The F145 or F146 instructions can only be executed if neither instruction is active. Evaluate the "F145/F146 not active" flag in your program to check the state of the instructions. The flag can be evaluated using one of the following instructions or system variables:

- **IsF145F146NotActive**
- **sys_bIsComPort1F145F146NotActive**
- **sys_bIsComPort2F145F146NotActive**

For communication ports not supporting the master function, the flag is always TRUE.

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iPort | INT | 0 |
| 1 | VAR | bF145F146NotActive | BOOL | FALSE |

### LD body

# 8 Communication instructions for networks

## 8.1 Number of connections

The number of connections for each communication is limited.

| Communication | Maximum number of connections |
|---|---|
| Ethernet communication | Max. 216 connections |
| EtherNet/IP communication | Max. 256 connections (including connections set with “I/O Map Setting” in Configurator EtherNet/IP) |
| UCMM message communication | Max. 256 connections |

Note

The total number of connections for the FP7 using Ethernet communication and EtherNet/IP communication should be max. 272.

## 8.2 FP instructions

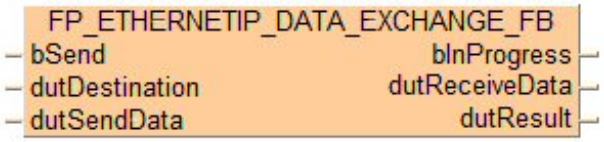
Tip

Advantages of FP instructions

# FP_ETHERNETIP_DATA_EXCHANGE_FB

Exchange message with EtherNet/IP unit

This function block exchanges an explicit message with an EtherNet/IP unit at the specified destination.




## Parameters

Input

**bSend** (BOOL)

Data exchange starts at a rising edge

**dutDestination** (**FP_ETHERNETIP_DATA_DESTINATION_DUT**)

Data unit type with information of the destination EtherNet/IP unit

**dutSendData** (**FP_ETHERNETIP_DATA_SEND_DUT**)

Data unit type of the send data buffer.
Data can be written into the send data buffer using the instruction **FP_ETHERNETIP_DATA_SET**

Output

**blnProgress** (BOOL)

Flag indicating that the data exchange with the destination EtherNet\IP unit is in progress.

**dutResult** (**FP_ETHERNETIP_DATA_EXCHANGE_RESULT_DUT**)

Data unit type of the instruction execution result

**dutReceiveData** (**FP_ETHERNETIP_DATA_RECEIVE_DUT**)

Data unit type of the receive data buffer.
Data can be read from the receive data buffer using the instructions **FP_ETHERNETIP_DATA_GET** and **FP_ETHERNETIP_DATA_GET_BYTES**

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

- if the EtherNet/IP function is not used in the Ethernet unit configuration.
- if the instruction is executed in an interrupt program
- if the size of the receive data buffer for **dutReceiveData** is outside the permissible range (3–253 words).
- if the size of the receive data buffer for **dutResult** is less than 3 words.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the EtherNet/IP function is not used in the Ethernet unit configuration.
- if the instruction is executed in an interrupt program
- if the size of the receive data buffer for **dutReceiveData** is outside the permissible range (3–253 words).
- if the size of the receive data buffer for **dutResult** is less than 3 words.

## Example

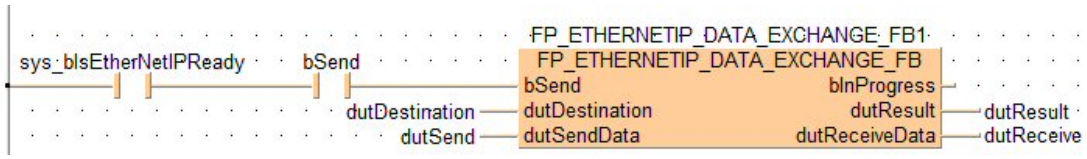
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | dutSend | FP_ETHERNETIP_DATA_SEND_DUT | |
| 2 | VAR | FP_ETHERNETIP_DATA_EXCHANGE_FB1 | FP_ETHERNETIP_DATA_EXCHANGE_FB | |
| 3 | VAR | bSend | BOOL | FALSE |
| 4 | VAR | dutDestination | FP_ETHERNETIP_DATA_DESTINATION_DUT | |
| 5 | VAR | dutReceive | FP_ETHERNETIP_DATA_RECEIVE_DUT | |
| 6 | VAR | dutResult | FP_ETHERNETIP_DATA_EXCHANGE_RESULT_DUT | |

### LD body

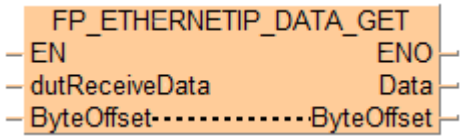
When the variables **sys_blsEtherNetIPReady** and **bSend** are set to TRUE, the function is executed.



# FP_ETHERNETIP_DATA_GET

Read data from receive buffer of EtherNet/IP unit

This FP instruction reads data from the receive data buffer specified by **FP_ETHERNETIP_DATA_RECEIVE_DUT** for an EtherNet/IP message that has been exchanged with **FP_ETHERNETIP_DATA_EXCHANGE_FB**.




## Parameters

Input

**dutReceiveData** (**FP_ETHERNETIP_DATA_RECEIVE_DUT**)

Data unit type of the receive data buffer.

Input/output

**ByteOffset** (WORD, INT, UINT)

- Current byte offset in the receive data buffer.
- For reading data from the starting address, set the byte offset 0.

Output

**Data** (WORD, INT, UINT)

- Data which will be read from the receive buffer.
- The amount of data which will be read from the receive buffer depends on the data type of the connected variable.
- For data type STRING, the current string length and the characters are read from the buffer.
- Boolean variables are not allowed.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the EtherNet/IP function is not used in the Ethernet unit configuration.
- if a value specified for a parameter is outside the permissible range.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the EtherNet/IP function is not used in the Ethernet unit configuration.
- if a value specified for a parameter is outside the permissible range.

## Example

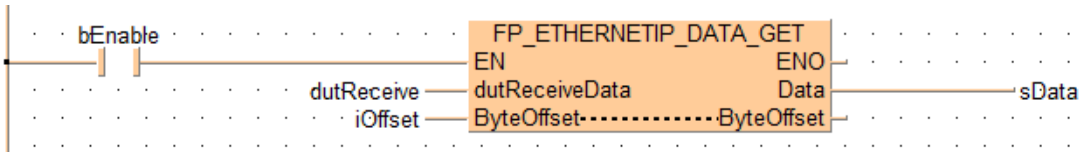
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | sData | STRING[32] | '' |
| 2 | VAR | iOffset | INT | 0 |
| 3 | VAR | bEnable | BOOL | FALSE |
| 4 | VAR | dutReceive | FP_ETHERNETIP_DATA_RECEIVE_DUT | |

### LD body

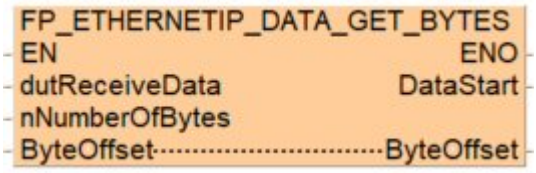
When the variable **bEnable** is set to TRUE, the function is executed.



# FP_ETHERNETIP_DATA_GET_BYTES

Read byte data from receive buffer of EtherNet/IP unit

This FP instruction reads byte data from the receive data buffer specified by **FP_ETHERNETIP_DATA_RECEIVE_DUT** for an EtherNet/IP message that has been exchanged with **FP_ETHERNETIP_DATA_EXCHANGE_FB**.




## Parameters

Input

**dutReceiveData** (**FP_ETHERNETIP_DATA_RECEIVE_DUT**)

Data unit type of the receive data buffer.

**nNumberOfBytes** (WORD, INT, UINT)

Number of bytes to be read from the receive data buffer.

Input/output

**ByteOffset** (WORD, INT, UINT)

Current byte offset in the receive data buffer.

Output

**DataStart** (WORD, INT, UINT)

Start address of the byte data to be read from the receive data buffer.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the EtherNet/IP function is not used in the Ethernet unit configuration.
- if a value specified for a parameter is outside the permissible range.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the EtherNet/IP function is not used in the Ethernet unit configuration.

- if a value specified for a parameter is outside the permissible range.

## Example

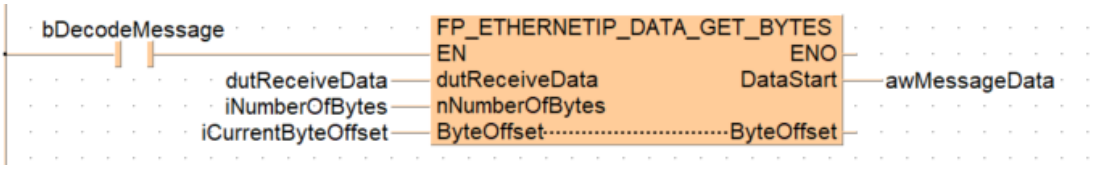
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | awMessageData | ARRAY [0..1] OF WORD | [2(16#0)] |
| 2 | VAR | dutReceiveData | FP_ETHERNETIP_DATA_RECEIVE_DUT | |
| 3 | VAR | iNumberOfBytes | INT | 3 |
| 4 | VAR | iCurrentByteOffset | INT | 0 |
| 5 | VAR | bDecodeMessage | BOOL | FALSE |

### LD body

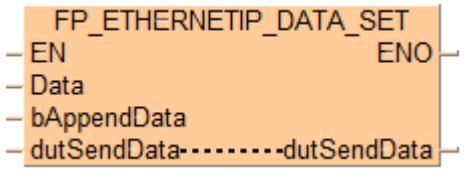
When the variable **bDecodeMessage** is set to TRUE, the function is carried out.



# FP_ETHERNETIP_DATA_SET

Write data to send buffer of EtherNet/IP unit

This FP instruction writes data into the send data buffer specified by **FP_ETHERNETIP_DATA_SEND_DUT** for an EtherNet/IP message that has been exchanged with **FP_ETHERNETIP_DATA_EXCHANGE_FB**.




## Parameters

Input

**Data** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

- Data which will be written into the send buffer.
- The amount of data which will be written into the send buffer depends on the data type of the connected variable.
- For data type STRING, the current string length and the characters are written into the send buffer.
- Boolean variables are not allowed.

**bAppendData** (BOOL)

Flag indicating that data should be appended to an existing buffer.

- When FALSE is set, the send buffer is cleared before writing data into the send buffer.
- When TRUE is set, the send buffer is not cleared and the data is appended to the data already in the buffer.

  The data is appended to the member **awData** at byte position **iDataSize_Bytes** of **FP_ETHERNETIP_DATA_SEND_DUT**.

Input/output

**dutSendData** (FP_ETHERNETIP_DATA_SEND_DUT)

Data unit type of the send data buffer.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the EtherNet/IP function is not used in the Ethernet unit configuration.

- if a value specified for a parameter is outside the permissible range.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the EtherNet/IP function is not used in the Ethernet unit configuration.
- if a value specified for a parameter is outside the permissible range.

## Example

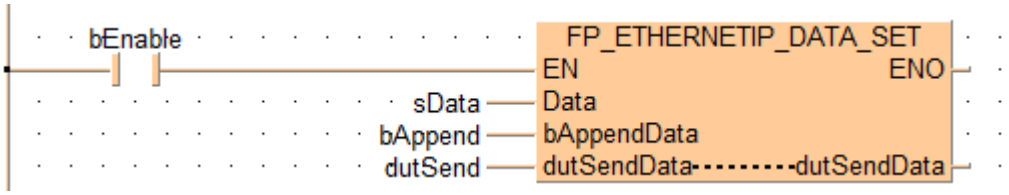
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bAppend | BOOL | FALSE |
| 2 | VAR | bEnable | BOOL | FALSE |
| 3 | VAR | sData | STRING[32] | '' |
| 4 | VAR | dutSend | FP_ETHERNETIP_DATA_SEND_DUT | |

### LD body

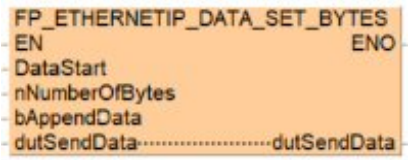
When the variable **bEnable** is set to TRUE, the function is executed.



# FP_ETHERNETIP_DATA_SET_BYTES

Write byte data to the send buffer of EtherNet/IP unit

This FP instruction writes byte data into the send data buffer specified by **FP_ETHERNETIP_DATA_SEND_DUT** for an EtherNet/IP message that has been exchanged with **FP_ETHERNETIP_DATA_EXCHANGE_FB**.




## Parameters

Input

**DataStart** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Start address of the byte data to be written to the send data buffer.

**nNumberOfBytes** (WORD, INT, UINT)

Number of bytes to be written to the send data buffer.

**bAppendData** (BOOL)

Flag indicating that data should be appended to an existing buffer.

- When FALSE is set, the send data buffer size is set to 0 and the data is written to the first position.
- When TRUE is set, data is written to the send data buffer at the current position and the send data buffer size is increased by **nNumberOfBytes**.

Input/output

**dutSendData** (**FP_ETHERNETIP_DATA_SEND_DUT**)

Data unit type of the send data buffer.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the EtherNet/IP function is not used in the Ethernet unit configuration.
- if a value specified for a parameter is outside the permissible range.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the EtherNet/IP function is not used in the Ethernet unit configuration.
- if a value specified for a parameter is outside the permissible range.

## Example

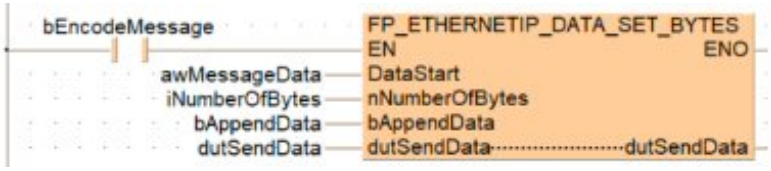
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | awMessageData | ARRAY [0..1] OF WORD | [16#0B0A,16#000C] |
| 2 | VAR | bEncodeMessage | BOOL | FALSE |
| 3 | VAR | bAppendData | BOOL | FALSE |
| 4 | VAR | dutSendData | FP_ETHERNETIP_DATA_SEND_DUT | |
| 5 | VAR | iNumberOfBytes | INT | 3 |

### LD body

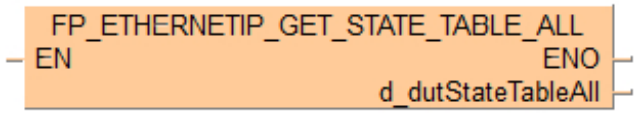
When the variable **bEncodeMessage** is set to TRUE, the function is carried out.



# FP_ETHERNETIP_GET_STATE_TABLE_ALL

Read the EtherNet/IP communication state on all nodes

This FP instruction reads the parameter information or status information on all registered nodes and stores it in the output variable of the type **FP_ETHERNETIP_STATE_TABLE_ALL_DUT**.




## Parameters

Output

**d_dutStateTableAll** (**FP_ETHERNETIP_STATE_TABLE_ALL_DUT**)

Destination to which the data is written

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

## Example

## POU header

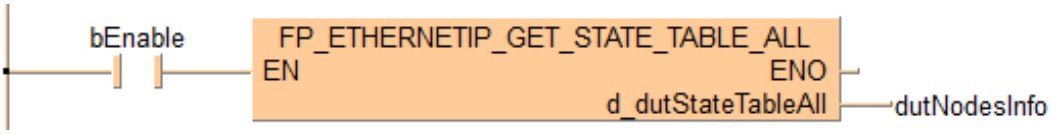
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | dutNodesInfo | FP_ETHERNETIP_STATE_TABLE_ALL_DUT | |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

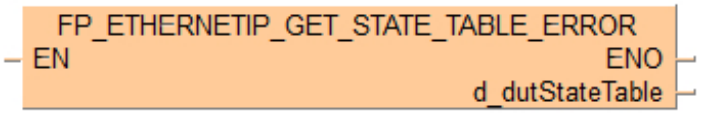
### LD body



# FP_ETHERNETIP_GET_STATE_TABLE_ERROR

Find all nodes with cyclic communication errors

This FP instruction reads the node number table to find all nodes with cyclic communication errors and stores the values in the output variable of the type **FP_ETHERNETIP_STATE_TABLE_DUT**.




## Parameters

Output

**d_dutStateTable** (**FP_ETHERNETIP_STATE_TABLE_DUT**)

Destination to which the data is written

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

## Example

## POU header

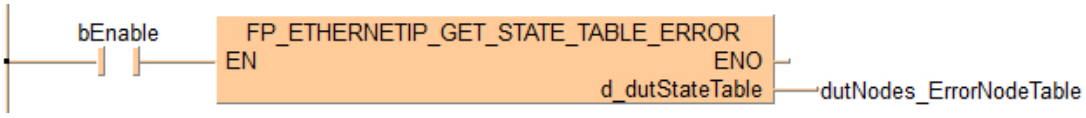
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | dutNodes_ErrorNodeTable | FP_ETHERNETIP_STATE_TABLE_DUT | |

### POU body

When the variable **bEnable** is set to TRUE, the function is executed.

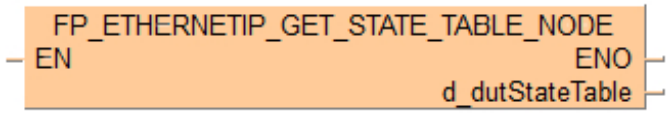
### LD body



# FP_ETHERNETIP_GET_STATE_TABLE_NODE

Find all nodes registered for cyclic communication

This FP instruction reads the node number table to find all nodes that have been registered for cyclic communication and stores the values in the output variable of the type **FP_ETHERNETIP_STATE_TABLE_DUT**.




## Parameters

Output

**d_dutStateTable** (**FP_ETHERNETIP_STATE_TABLE_DUT**)

Destination to which the data is written

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

## Example

## POU header

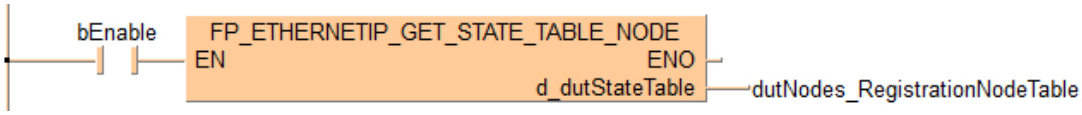
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | dutNodes_RegistrationNodeTable | FP_ETHERNETIP_STATE_TABLE_DUT | |

### POU body

When the variable **bEnable** is set to TRUE, the function is executed.

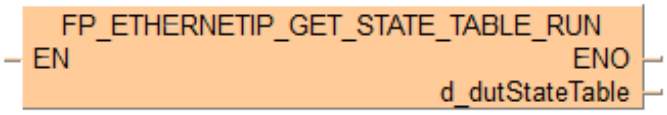
### LD body



# FP_ETHERNETIP_GET_STATE_TABLE_RUN

Find all nodes with normal cyclic communication

This FP instruction reads the node number table to find all nodes where the cyclic communication runs normally and stores the values in the output variable of the type **FP_ETHERNETIP_STATE_TABLE_DUT**.




## Parameters

Output

**d_dutStateTable** (**FP_ETHERNETIP_STATE_TABLE_DUT**)

Destination to which the data is written

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

## Example

## POU header

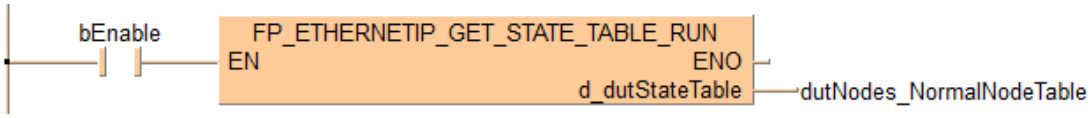
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | dutNodes_NormalNodeTable | FP_ETHERNETIP_STATE_TABLE_DUT | |

### POU body

When the variable **bEnable** is set to TRUE, the function is executed.

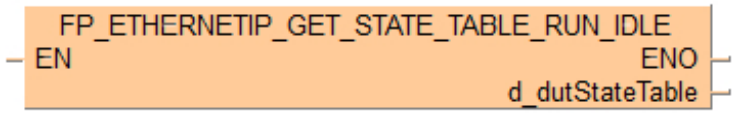
### LD body



# FP_ETHERNETIP_GET_STATE_TABLE_RUN_IDLE

Check RUN/IDLE bit for state of cyclic communication

This FP instruction checks the state of the cyclic communication by reading the RUN/IDLE bit of each registered node from the node number table and stores the values in the output variable of the type **FP_ETHERNETIP_STATE_TABLE_DUT**.




## Parameters

Output

**d_dutStateTable** (**FP_ETHERNETIP_STATE_TABLE_DUT**)

Destination to which the data is written

## Remarks

Bit corresponding to the node number of FP7 whose connection is registered. When the following two conditions are met, it turns to TRUE, in other conditions, it turns to FALSE.

- Communication with the target node (FP7) runs normally.
- Communication with all nodes except FP7 runs normally when the target node (FP7) is in “RUN mode”.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

## Example

### POU header

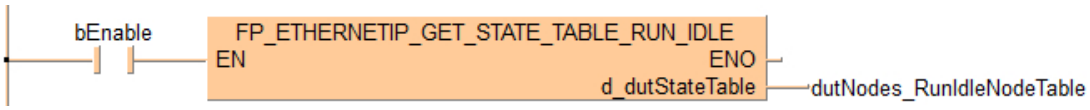
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | dutNodes_RunIdleNodeTable | FP_ETHERNETIP_STATE_TABLE_DUT | |

### POU body

When the variable **bEnable** is set to TRUE, the function is executed.

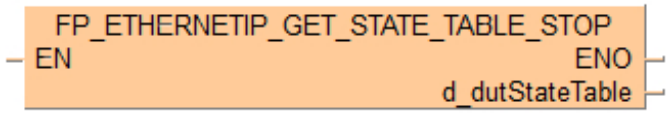
### LD body



# FP_ETHERNETIP_GET_STATE_TABLE_STOP

Find all nodes where cyclic communication has stopped

This FP instruction reads the node number table to find all nodes where the cyclic communication has stopped and stores the values in the output variable of the type **FP_ETHERNETIP_STATE_TABLE_DUT**.




## Parameters

Output

**d_dutStateTable** (**FP_ETHERNETIP_STATE_TABLE_DUT**)

Destination to which the data is written

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

## Example

## POU header

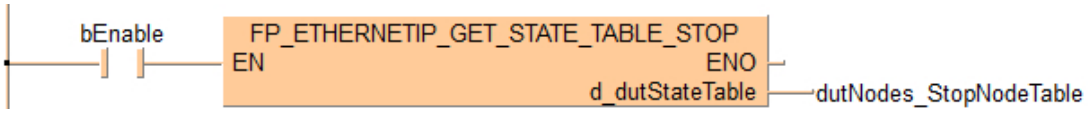
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | dutNodes_StopNodeTable | FP_ETHERNETIP_STATE_TABLE_DUT | |

### POU body

When the variable **bEnable** is set to TRUE, the function is executed.

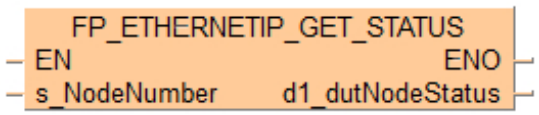
### LD body



# FP_ETHERNETIP_GET_STATUS

Read the status of a node

This FP instruction reads the status of the node specified by **s_NodeNumber** and stores the values in the output variable of the type **FP_ETHERNETIP_NODE_STATUS_DUT**.




## Parameters

Input

**s_NodeNumber** (WORD, INT, UINT)

Node number

Output

**d1_dutNodeStatus** (**FP_ETHERNETIP_NODE_STATUS_DUT**)

Destination to which the data is written

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

## Example

## POU header

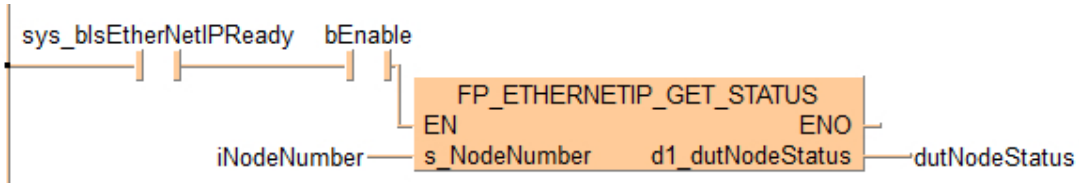
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | iNodeNumber | INT | 0 |
| 2 | VAR | dutNodeStatus | FP_ETHERNETIP_NODE_STATUS_DUT | |

## POU body

When the variables **sys_blsEtherNetIPReady** and **bEnable** are set to TRUE, the function is executed.

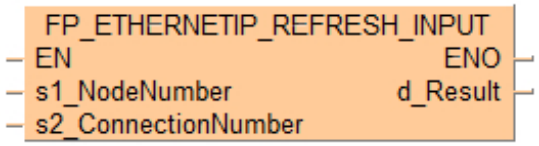
## LD body



# FP_ETHERNETIP_REFRESH_INPUT

Refresh EtherNet/IP input

This FP instruction refreshes the input data from the receive buffer of the node and connection number specified. Data are copied from the PLC's receive buffer to the PLC's operation memory.




## Parameters

Input

**s1_NodeNumber** (WORD, INT, UINT)

Node number of the input data to refresh. Scan list range: 1–256

- An error occurs when the value specified is outside the scan list range.
- An error also occurs when a reserved node is specified.

**s2_ConnectionNumber** (WORD, INT, UINT)

Node connection number of the input data to refresh. Range: 1–256
An error occurs when the value specified is outside the scan list range.

Output

**d_Result** (WORD)

Memory area storing the refresh results.

- 0: Refresh operation is completed successfully.
- 1: No data is received. Refresh is not performed.
- 2: EtherNet/IP is not ready for communication.

## Remarks

- Before you execute the instruction, check if the communication of a specified connection runs normally using **FP_ETHERNETIP_GET_STATE_TABLE_RUN**.
- Call this instruction after **sys_bIsEtherNetIPReady** turns to TRUE. If it is called before **sys_bIsEtherNetIPReady** turns to TRUE, an error indicating EtherNet/IP is not ready for communication is returned.
- Do not execute this instruction continuously in one scan to reduce communication load.

- Use this instruction only for the connection for which “Instruction” is selected as “Refresh Method” in “EtherNet/IP setting”.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the EtherNet/IP function is not used in the Ethernet unit configuration.
- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if the node or connection specified by **s1_NodeNumber** and **s2_ConnectionNumber** does not exist.
- if the refresh method selected for the connection is something other than “Instruction”.
- if the number of input data of the specified connection is 0.
- if the number of refreshed data of the specified connection is 0.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the EtherNet/IP function is not used in the Ethernet unit configuration.
- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if the node or connection specified by **s1_NodeNumber** and **s2_ConnectionNumber** does not exist.
- if the refresh method selected for the connection is something other than “Instruction”.
- if the number of input data of the specified connection is 0.
- if the number of refreshed data of the specified connection is 0.

## Example

## POU header

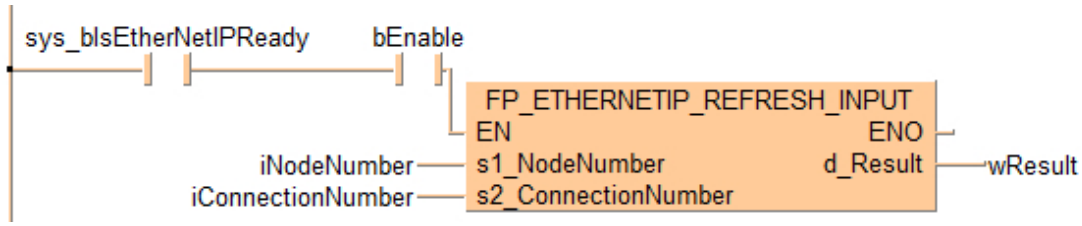
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | iNodeNumber | INT | 0 |
| 2 | VAR | iConnectionNumb... | INT | 0 |
| 3 | VAR | wResult | WORD | 0 |

### POU body

When the variables **sys_bIsEtherNetIPReady** and **bEnable** are set to TRUE, the function is executed.

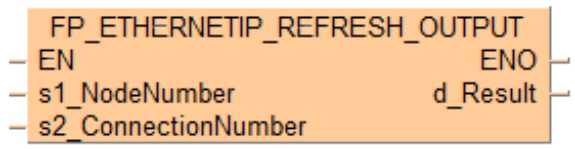
### LD body



# FP_ETHERNETIP_REFRESH_OUTPUT

Refresh EtherNet/IP output

This FP instruction refreshes the output data for the node and connection number specified. Data are copied from the PLC's operation memory to the PLC's send buffer.




## Parameters

Input

**s1_NodeNumber** (WORD, INT, UINT)

Node number of the output data to refresh. Scan list range: 1–256
An error occurs when the value specified is outside the scan list range.
The I/O map is used for sending data to a target device (PLC).

**s2_ConnectionNumber** (WORD, INT, UINT)

Connection number of the output data to refresh. Range: 1–256
An error occurs when the value specified is outside the scan list range.

Output

**d_Result** (WORD)

Memory area storing the refresh results. If this instruction is executed in cycles faster than RPI, the output refresh may not be performed.

- 0: Refresh operation is completed successfully.
- 1: No data is received. Refresh is not performed.
- 2: EtherNet/IP is not ready for communication.

## Remarks

- Before you execute the instruction, check if the communication of a specified connection runs normally using **FP_ETHERNETIP_GET_STATE_TABLE_RUN**.
- Call this instruction after **sys_bIsEtherNetIPReady** turns to TRUE. If it is called before **sys_bIsEtherNetIPReady** turns to TRUE, an error indicating EtherNet/IP is not ready for communication is returned.
- Do not execute this instruction continuously in one scan to reduce communication load.
- Use this instruction only for the connection for which "Instruction" is selected as "Refresh Method" in "EtherNet/IP setting".

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the EtherNet/IP function is not used in the Ethernet unit configuration.
- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if the I/O map or the node/connection specified by **s1_NodeNumber** and **s2_ConnectionNumber** does not exist.
- if the refresh method selected for the connection is something other than “Instruction”.
- if the number of output data of the specified connection is 0.
- if the number of refreshed data of the specified connection is 0.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the EtherNet/IP function is not used in the Ethernet unit configuration.
- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if the I/O map or the node/connection specified by **s1_NodeNumber** and **s2_ConnectionNumber** does not exist.
- if the refresh method selected for the connection is something other than “Instruction”.
- if the number of output data of the specified connection is 0.
- if the number of refreshed data of the specified connection is 0.

## Example

### POU header

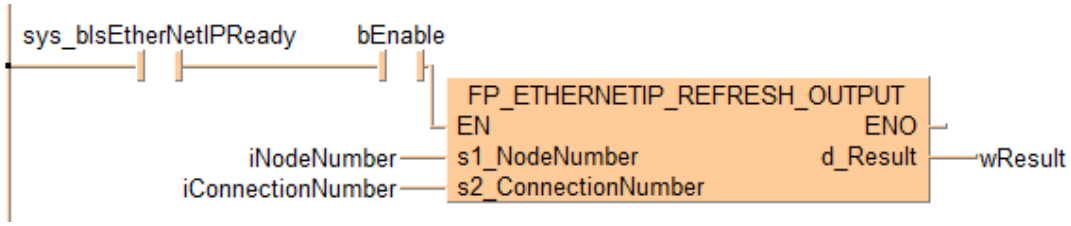
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | iNodeNumber | INT | 0 |
| 2 | VAR | iConnectionNumb... | INT | 0 |
| 3 | VAR | wResult | WORD | 0 |

### POU body

When the variables **sys_bIsEtherNetIPReady** and **bEnable** are set to TRUE, the function is executed.

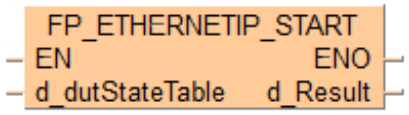
### LD body


sys_blsEtherNetIPReady
bEnable
FP_ETHERNETIP_REFRESH_OUTPUT
EN
ENO
iNodeNumber
s1_NodeNumber
d_Result
wResult
iConnectionNumber
s2_ConnectionNumber

# FP_ETHERNETIP_START

Cyclic communication start request

This FP instruction starts the node on which the start request is made and writes the result into the output variable **d_Result**. The node number table is specified by **FP_ETHERNETIP_STATE_TABLE_DUT**.




## Parameters

Input

**d_dutStateTable** (**FP_ETHERNETIP_STATE_TABLE_DUT**)

- Node number table for start requests
- Memory area address storing the maximum node number (1-256) or a constant.

Output

**d_Result** (WORD)

Destination to which the data is written

- 0: Specified node start is completed
- 1: Specified node start processing is in progress.
- 2: Specified node start failed
- 3: Multiple starts of **FP_ETHERNETIP_START**

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

## Example

### POU header

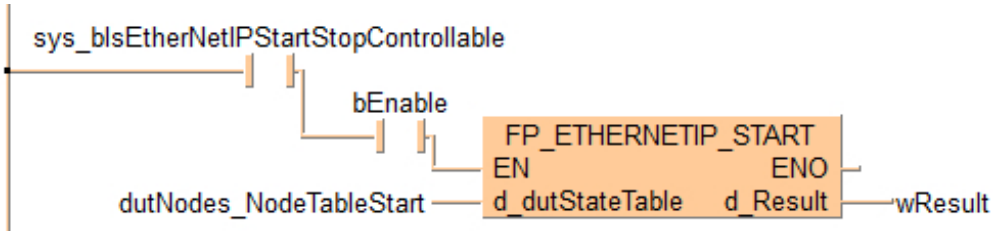
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | wResult | WORD | 0 |
| 2 | VAR | dutNodes_NodeTableStart | FP_ETHERNETIP_STATE_TABLE_DUT | |

### POU body

When the variables **sys_blsEtherNetIPStartStopControllable** and **bEnable** are set to TRUE, the function is carried out.

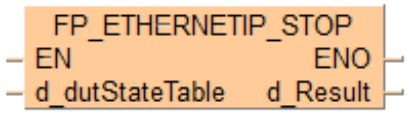
### LD body



# FP_ETHERNETIP_STOP

Cyclic communication stop request

This FP instruction stops the node on which the stop request is made and writes the result into the output variable **d_Result**. The node number table is specified by **FP_ETHERNETIP_STATE_TABLE_DUT**.




## Parameters

Input

**d_dutStateTable** (**FP_ETHERNETIP_STATE_TABLE_DUT**)

- Node number table for stop requests
- Memory area address storing the maximum node number (1-256) or a constant.

Output

**d_Result** (WORD)

Destination to which the data is written

- 0: Specified node stop is completed
- 1: Specified node stop processing is in progress.
- 2: Specified node stop failed
- 3: Multiple starts of **FP_ETHERNETIP_STOP**

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the read area is outside the permissible range
- if the instruction is executed in an interrupt program

## Example

### POU header

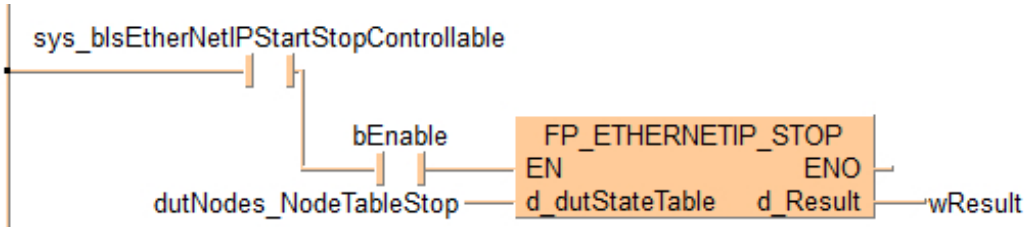
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | wResult | WORD | 0 |
| 2 | VAR | dutNodes_NodeTableStart | FP_ETHERNETIP_STATE_TABLE_DUT | |

### POU body

When the variables **sys_blsEtherNetIPStartStopControllable** and **bEnable** are set to TRUE, the function is carried out.

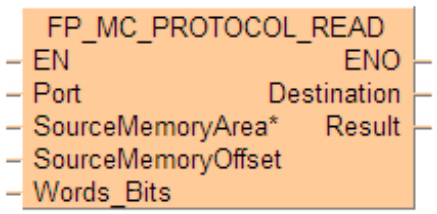
### LD body



# FP_MC_PROTOCOL_READ

Read data from external devices via MC protocol

The PLC has the transmission right in the master communication. The communication is performed by sending commands to devices which support MC protocol and receiving responses. By specifying a memory address and executing a read instruction in a user program, the PLC generates a message according to the protocol automatically.




## Parameters

Input

**Port** (WORD, INT, UINT)

Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**

**SourceMemoryArea** (WORD, INT, UINT)

Memory area on the slave from where to request the data.

| Unit | Memory area type | System variable | |
|---|---|---|---|
| Bit | Input | X | Hexadecimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_INPUT_X** |
| | Output | Y | Hexadecimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_OUTPUT_Y** |
| | Link relay | B | Hexadecimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_LINK_RELAY_B** |
| | Internal relay | M | Decimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_INTERNAL_RELAY_M** |
| | Latch relay | L | Decimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_LATCH_RELAY_L** |
| Word | Data register | D | Decimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_DATA_REGISTER_D** |
| | File register | R | Decimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_FILE_REGISTER_R** |

| Unit | Memory area type | System variable | |
|---|---|---|---|
| | | ZR | Hexadecimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_FILE_REGISTER_ZR** |
| | Link register | W | Hexadecimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_LINK_REGISTER_W** |

**SourceMemoryOffset** (DWORD, DINT, UDINT)

Offset of the memory area on the slave from where to request the data.
Range: 16#0–16#F7FFFF (0–16252927)

**Words_Bits** (WORD, INT, UINT)

The transfer method varies according to the data type of **Destination**.

| Memory area specified in Destination | Transfer method | No. of transmitted data | Remarks |
|---|---|---|---|
| 16-bit memory area WX, WY, WR, WL, DT, LD | Word transmission | 1–960 words | |
| 1-bit memory area X,Y,R,L,DT,n,LD,n | Bit transmission | 1–7168 bits | When the number of transmitted data is odd, 4-bit dummy code 16#0 is added. |

Output

**Destination** (ANY)

Word area or register on the master unit to which the requested data is written.

**Result** (WORD, INT, UINT)

For FP7 only:

- 0: Normal completion
- 1: Communication port is used for master communication
- 2: Communication port is used for slave communication
- 3: No. of master communication instructions that can be used at the same time is exceeded
- 4: Transmission timeout
- 5: Timeout while waiting for a response
- 6: Error in received data

  This error occurs when an abnormal telegram is received, e.g. when there is a format error in the header. In this case, the received data is discarded.
- 7: Insufficient area reserved in I/O map

This error occurs when the number of Ethernet connections exceeds 16 (Advanced Ethernet communication). Check whether the word area reserved for communication control flags is big enough for the number of Ethernet connections.

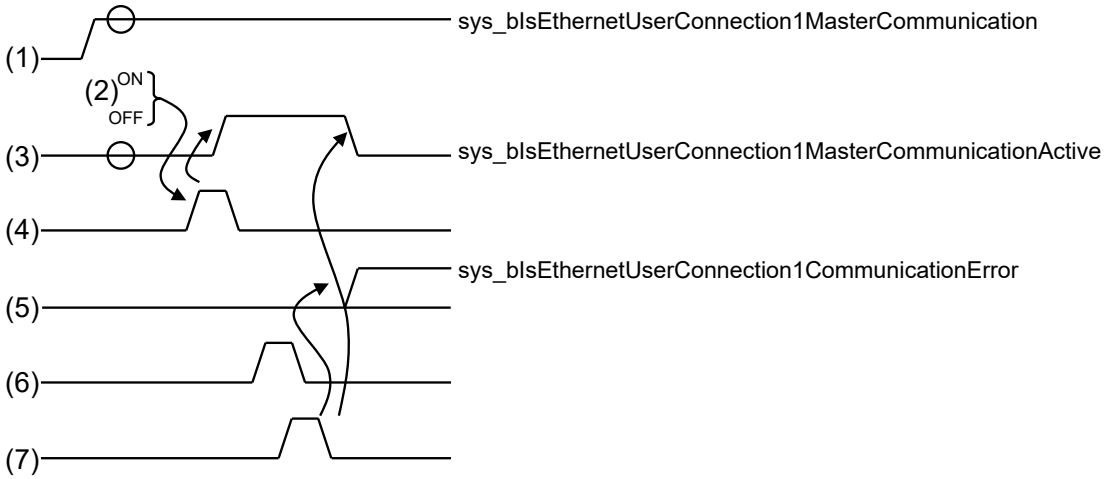
- 8: Send buffer currently in use

  This error may occur for FP7 PLCs with firmware version 4.57 or later.

In case of communication errors, other error codes specific to the MC protocol may be set. For more details please refer to the documentation of the PLC used.

For other PLCs: set to 0

## Time chart




(1) Master communication clear-to-send flag,e.g.
**sys_bIsEthernetUserConnection1MasterCommunication**

(2) Check that the master communication clear-to-send flag is TRUE and check that the master communication sending flag is FALSE

(3) Master communication sending flag, e.g.
**sys_bIsEthernetUserConnection1MasterCommunicationActive**
While sending: Master communication sending flag is TRUE
Sending done: Master communication sending flag is FALSE

(4) Execute this instruction

(5) Master communication sending done flag, e.g.
**sys_bIsEthernetUserConnection1CommunicationError**
Normal completion: FALSE
Abnormal completion: TRUE

(6) Send data

(7) Processing of the received response

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- If slave or master data exceed the available address range.
- if the Ethernet port is incorrect or the Ethernet connection is closed.
- if the number of sent data specified by **Words_Bits** is incorrect.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- If slave or master data exceed the available address range.
- if the Ethernet port is incorrect or the Ethernet connection is closed.
- if the number of sent data specified by **Words_Bits** is incorrect.

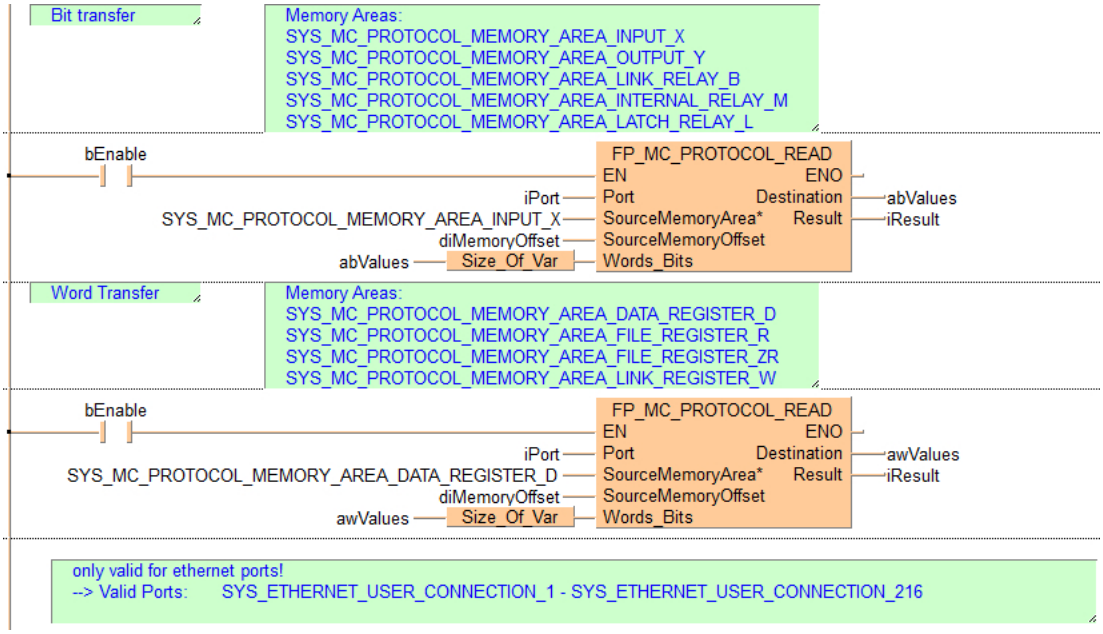
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | diMemoryOffset | DINT | 0 |
| 1 | VAR | awValues | ARRAY [0..2] OF WORD | [3(0)] |
| 2 | VAR | iResult | INT | 0 |
| 3 | VAR | bEnable | BOOL | FALSE |
| 4 | VAR | abValues | ARRAY [0..2] OF BOOL | [3(FALSE)] |
| 5 | VAR | iPort | INT | 0 |

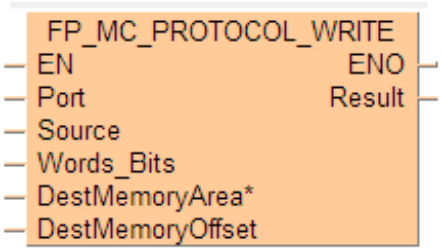
## LD body



# FP_MC_PROTOCOL_WRITE

Write data to external devices via MC protocol

The PLC has the transmission right in the master communication. The communication is performed by sending commands to devices which support MC protocol and receiving responses. By specifying a memory address and executing a write instruction in a user program, the PLC generates a message according to the protocol automatically.




## Parameters

Input

**Port** (WORD, INT, UINT)

Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**

**Source** (ANY)

Word area or register on the master unit for the data to be written to the slave.

**Words_Bits** (WORD, INT, UINT)

The transfer method varies according to the data type of **Source**.

| Memory area specified in Source | Transfer method | No. of transmitted data | Remarks |
|---|---|---|---|
| 16-bit memory area WX, WY, WR, WL, DT, LD | Word transmission | 1–960 words | |
| 1-bit memory area X,Y,R,L,DT,n,LD,n | Bit transmission | 1–7168 bits | When the number of transmitted data is odd, 4-bit dummy code 16#0 is added. |

**DestMemoryArea** (WORD, INT, UINT)

Memory area on the slave to which the data is written.

| Unit | Memory area type | System variable | |
|---|---|---|---|
| Bit | Input | X | Hexadecimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_INPUT_X** |
| | Output | Y | Hexadecimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_OUTPUT_Y** |
| | Link relay | B | Hexadecimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_LINK_RELAY_B** |
| | Internal relay | M | Decimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_INTERNAL_RELAY_M** |
| | Latch relay | L | Decimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_LATCH_RELAY_L** |
| Word | Data register | D | Decimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_DATA_REGISTER_D** |
| | File register | R | Decimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_FILE_REGISTER_R** |
| | | ZR | Hexadecimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_FILE_REGISTER_ZR** |
| | Link register | W | Hexadecimal<br>**SYS_MC_PROTOCOL_MEMORY_AREA_LINK_REGISTER_W** |

**DestMemoryOffset** (DWORD, DINT, UDINT)

Offset of the memory area on the slave to which the data is written.
Range: 16#0–16#F7FFFF (0–16252927)

Output

**Result** (WORD, INT, UINT)

For FP7 only:

- 0: Normal completion
- 1: Communication port is used for master communication
- 2: Communication port is used for slave communication
- 3: No. of master communication instructions that can be used at the same time is exceeded
- 4: Transmission timeout
- 5: Timeout while waiting for a response
- 6: Error in received data

  This error occurs when an abnormal telegram is received, e.g. when there is a format error in the header. In this case, the received data is discarded.
- 7: Insufficient area reserved in I/O map

  This error occurs when the number of Ethernet connections exceeds 16 (Advanced Ethernet communication). Check whether the word area reserved for communication control flags is big enough for the number of Ethernet connections.

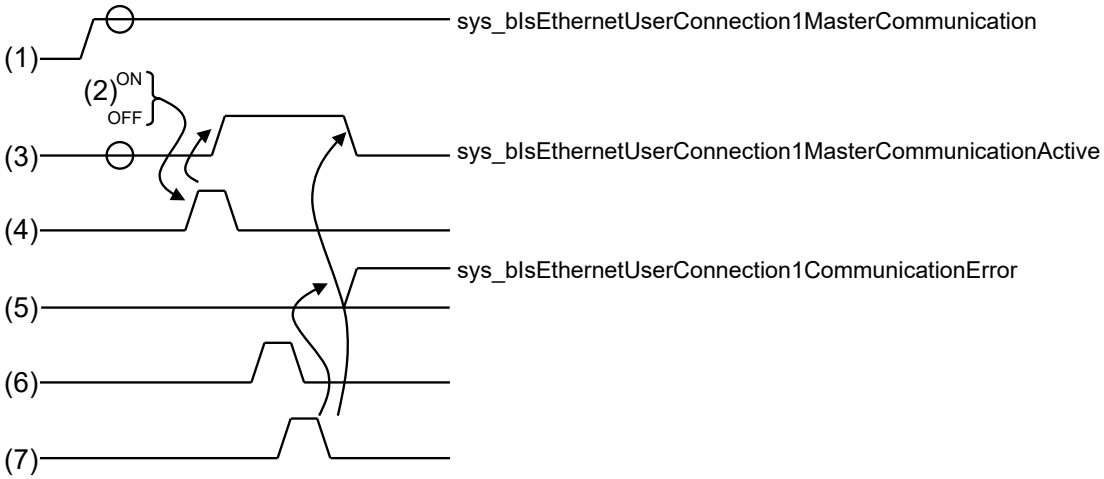
- 8: Send buffer currently in use

  This error may occur for FP7 PLCs with firmware version 4.57 or later.

In case of communication errors, other error codes specific to the MC protocol may be set. For more details please refer to the documentation of the PLC used.

For other PLCs: set to 0

## Time chart




(1) Master communication clear-to-send flag,e.g. **sys_bIsEthernetUserConnection1MasterCommunication**

(2) Check that the master communication clear-to-send flag is TRUE and check that the master communication sending flag is FALSE

(3) Master communication sending flag, e.g. **sys_bIsEthernetUserConnection1MasterCommunicationActive**
While sending: Master communication sending flag is TRUE
Sending done: Master communication sending flag is FALSE

(4) Execute this instruction

(5) Master communication sending done flag, e.g. **sys_bIsEthernetUserConnection1CommunicationError**
Normal completion: FALSE
Abnormal completion: TRUE

(6) Send data

(7) Processing of the received response

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- If slave or master data exceed the available address range.
- if the Ethernet port is incorrect or the Ethernet connection is closed.
- if the number of sent data specified by **Words_Bits** is incorrect.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- If slave or master data exceed the available address range.

- if the Ethernet port is incorrect or the Ethernet connection is closed.
- if the number of sent data specified by **Words_Bits** is incorrect.

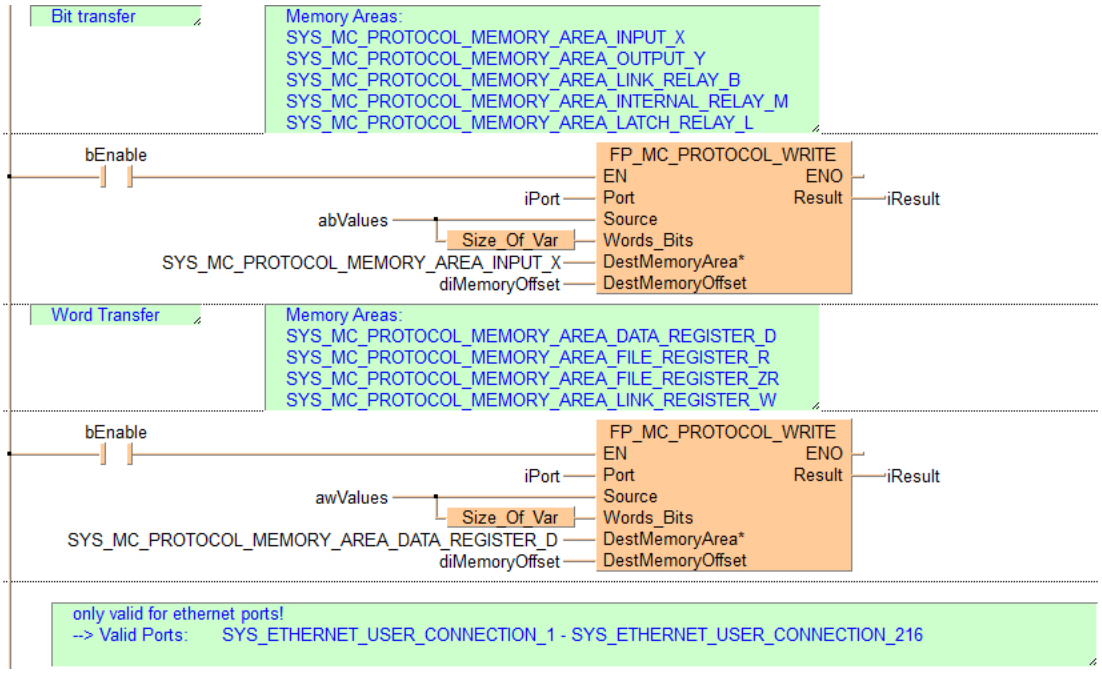
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | awValues | ARRAY [0..2] OF WORD | [3(0)] |
| 2 | VAR | diMemoryOffset | DINT | 0 |
| 3 | VAR | iResult | INT | 0 |
| 4 | VAR | abValues | ARRAY [0..2] OF BOOL | [3(FALSE)] |
| 5 | VAR | iPort | INT | 0 |

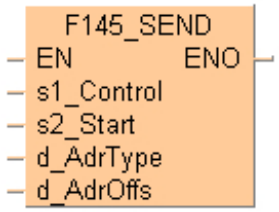
### LD body



# F145_SEND

Send data (MEWNET link)

This instruction sends data to another station through link modules in the network.




## Parameters

Input

**s1_Control** (DWORD)

32-bit area for storing control data

**s2_Start** (WORD, INT, UINT)

Starting 16-bit area for storing source data (data area at the source station)

**d_AdrType** (WORD, INT, UINT)

Type of destination operands for storing data in the target station. Be sure to select the area by setting the address 0 (e.g. DT0 or WR0, ...) (data area at the target station)

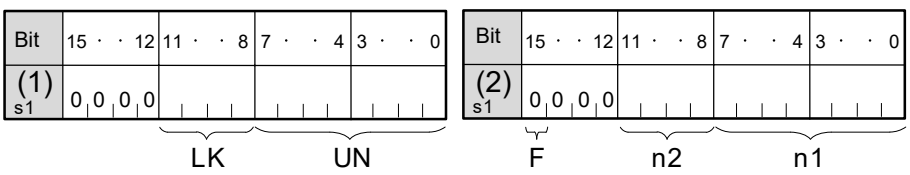
**d_AdrOffs** (WORD, INT, UINT)

Starting 16-bit area address for the destination operand specified in **d_AdrType** (data area at the target station), must be a constant

The variables **s2_Start**, **d_AdrType** and **d_AdrOffs** have to be of the same data type.

## Remarks

Specifications of **s1_Control**:

| **s1 higher byte** | **s1 lower byte** |
|---|---|
|  | |
| 1. Link No. selection (LK: 1–3, the station itself) | 1. Word unit send selection |

| s1 higher byte | s1 lower byte | |
|---|---|---|
| Up to 3 link units can be connected to 1 CPU. | F = 0 | Word unit selection |
| | n2 = 0 | Set "0" when the word unit is selected |
| This (LK) selects the source link unit of the three. | n1 = 11–16 | Specify the number of words to be sent |
| 2. Link station No. selection (UN: 1–63, another station) | 2. Bit unit send selection | |
| Up to 63 stations can be connected to 1 link unit. | F = 1 | Bit unit selection |
| This (UN) then selects the target station No. | n2 = 0–15 | Destination bit No. |
| | n1 = 0–15 | Source bit No. |

Tip

For detailed information, please refer to the relevant technical manual of the intelligent unit.

## Example

### POU header

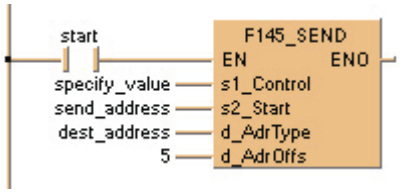
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | specify_value | DWORD | 0 | stores the control data |
| 2 | VAR | send_address | WORD | 0 | Starting 16-bit area for |
| 3 | VAR | dest_address | WORD | 0 | Type of destination |
| 4 | VAR | n | INT | 0 | operands for storing data |
| 5 | VAR | | | | in the destination station |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

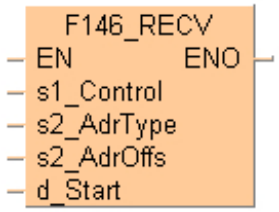
### LD body



# F146_RECV

Receive data (MEWNET link)

This instruction receives data from another station through link units in the network.




## Parameters

Input

**s1_Control** (DWORD)

32-bit area for storing control data

**s2_AdrType** (WORD, INT, UINT)

Type of source operands for storing data in the target station. Be sure to select the area by setting the address 0 (e.g. DT0 or WR0, ...) (data area at the source station)

**s2_AdrOffs** (WORD, INT, UINT)

Starting 16-bit area address for the source operand specified in **s2_AdrType** (data area at the source station)

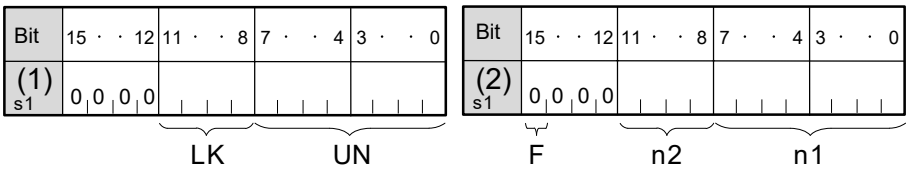
**d_Start** (WORD, INT, UINT)

Starting 16-bit area address for storing data received (data area at the target station), must be a constant

The variables **s2_AdrType**, **s2_AdrOffs** and **d_Start** have to be of the same data type.

## Remarks

Specifications of **s1_Control**:

| s1 higher byte | | s1 lower byte | |
|---|---|---|---|
|  | | | |
| 1. Link No. selection (LK: 1–3, the station itself) | | 1. Word unit send selection | |
| Up to 3 link units can be connected to 1 CPU. | | F = 0 | Word unit selection |
| | | n2 = 0 | Set "0" when the word unit is selected |
| This (LK) selects the source link unit of the three. | | n1 = 11–16 | Specify the number of words to be sent |
| 2. Link station No. selection (UN: 1–63, another station) | | 2. Bit unit send selection | |
| Up to 63 stations can be connected to 1 link unit. | | F = 1 | Bit unit selection |
| This (UN) then selects the target station No. | | n2 = 0–15 | Destination bit No. |
| | | n1 = 0–15 | Source bit No. |

Tip

For detailed information, please refer to the relevant technical manual of the intelligent unit.

## Example

### POU header

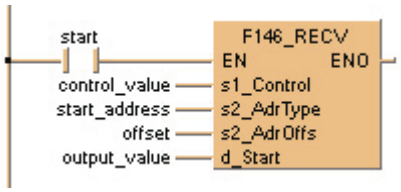
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | control_value | DWORD | 0 | 32-bit area for storing control data |
| 2 | VAR | start_address | WORD | 0 | Starting 16-bit area address for |
| 3 | VAR | output_value | WORD | 0 | Starting 16-bit area address for storing data received |
| 4 | VAR | | | | |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

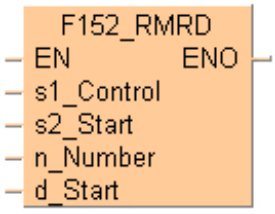
## LD body


start
F146_RECV
EN
ENO
control_value
s1_Control
start_address
s2_AdrType
offset
s2_AdrOffs
output_value
d_Start

# F152_RMRD

Read data from the slave station

This instruction reads data from the specified intelligent unit of the MEWNET-F slave station.




## Parameters

Input

**s1_Control** (DWORD)

Stores control data for master/slave configuration

**s2_Start** (WORD, INT, UINT)

Starting address of the memory area to be read

**n_Number** (INT)

Number of words to be read (max. 32 words)

**d_Start** (WORD, INT, UINT)

Starting 16-bit area where words read are stored, (see F153)

The variables **s2_Start** and **d_Start** have to be of the same data type.

## Remarks

- It is not possible to execute multiple **F152_RMRD** instructions and **F153_RMWT** instructions at the same time. The program should be set up so that these instructions are executed when the **F152_RMRD**/**F153_RMWT** instruction execution enabled flag **sys_bIsMewnetFNotActive** is TRUE.

| | |
|---|---|
| **sys_bIsMewnetFNotActive** | 0: Execution inhibited (**F152_RMRD**/**F153_RMWT** instruction being executed)<br>1: Execution enabled |

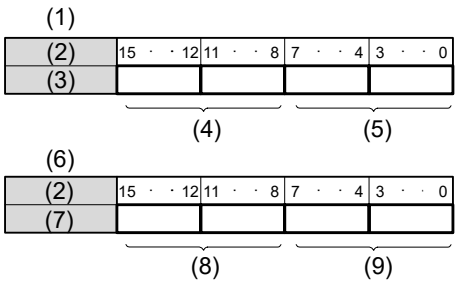
- The **F152_RMRD** instruction only enables a request to be accepted. The actual processing is carried out at the end of the scan. The **F152_RMRD**/**F153_RMWT** instruction completed flag (R9036) can be used to confirm whether or not the instruction has been executed.

| | |
|---|---|
| **sys_bIsMewnetFError** | 0: Normal completion<br>1: Completed with error (the error code is stored in **sys_wMewnetFErrorCode**) |
| **sys_wMewnetFErrorCode** | If the transmission has been completed with an error (**sys_bIsMewnetFError** (turns to TRUE and remains TRUE) , the contents of the error (error code) are stored. |

| Error code (HEX) | Description |
|---|---|
| 16#5B | Timeout error (no intelligent unit found at the specified location) |
| 16#68 | No memory error (no memory exists at the specified address) |
| 16#71 | Send answer timeout error |
| 16#72 | Send buffer full timeout error |
| 16#73 | Response timeout error |

- If the error code is 16#71–16#73, a communication timeout error has occurred. The timeout time can be changed within a range of 10.0ms to 81.9s (in units of 10ms), using the setting of system register 32. The default value is set to 2 seconds.
- **s1** stores the control data for the configuration of the master and slave units in the network. **n** words are read beginning from the shared memory address number in the intelligent unit specified by **s2_Start**. The result is stored in **d**.

  Specifications of **s1_Control**:




(1) **s1** higher word
(2) Bit
(3) **s1** higher word
(4) Bank No. (16#00–16#FF if there is a bank to specify, otherwise 16#00)
(5) Slot No. (16#00–16#1F, FP3: 16#00–16#17)
(6) **s1** lower word
(7) **s1** lower word
(8) Master station No.(16#01–16#04)
(9) Slave station No. (16#01–16#20)

Tip

Intelligent unit with bank: FP3 expansion data memory unit

Order number: AFP32091 AFP32092

## Error flags

**sys_bIsOperationErrorHold**

- if the control data **s1_Control** exceeds the limit of specified range
- if no MEWNET-F master unit is found
- if the data read exceeds the area of **s2_Start**

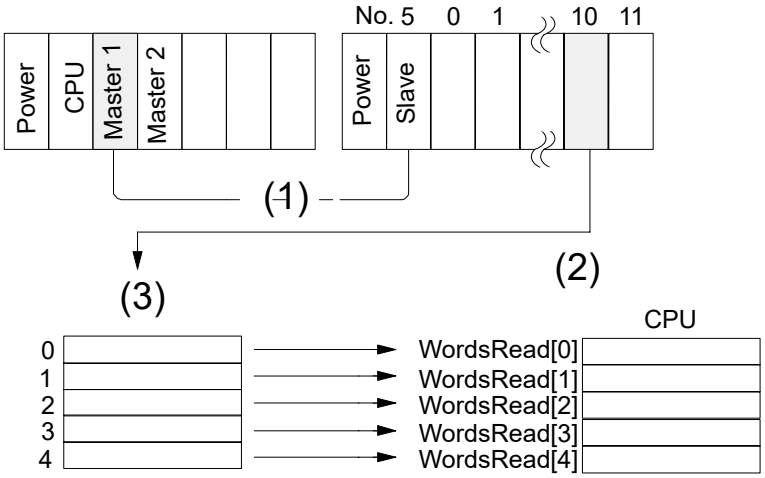
**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the control data **s1_Control** exceeds the limit of specified range
- if no MEWNET-F master unit is found
- if the data read exceeds the area of **s2_Start**

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.



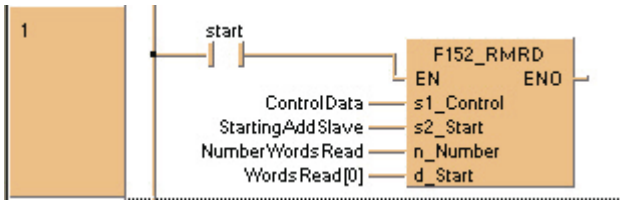

(1) Master station 1

(2) Control data = 16#A0105

(3) Intelligent unit (shared memory)

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | Start | BOOL | FALSE | |
| 1 | VAR | Control Data | DWORD | 16#000A0105 | No bank, slot no. 10, Master station 1, Slave station 5 |
| 2 | VAR | StartingAddSlave | WORD | 0 | |
| 3 | VAR | NumberWordsRead | INT | 5 | |
| 4 | VAR | WordsRead | ARRAY [0..4] OF WORD | | |

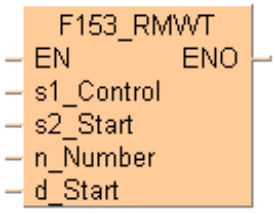
## LD body


1
start
F152_RMRD
EN
ENO
ControlData
s1_Control
StartingAddSlave
s2_Start
NumberWordsRead
n_Number
WordsRead[0]
d_Start

# F153_RMWT

Write data into the slave station

This instruction writes data into the specified intelligent unit of the MEWNET-F slave station.




## Parameters

Input

**s1_Control** (DWORD)

Stores control data for master/slave configuration

**s2_Start** (WORD, INT, UINT)

Starting 16-bit area in CPU where words are read

**n_Number** (INT)

Number of words to be read and then written to the slave unit (max. 32 words)

**d_Start** (WORD, INT, UINT)

Starting memory address number in the intelligent unit where words are written

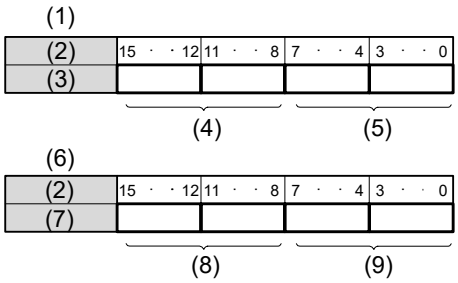
The variables **s2_Start** and **d_Start** have to be of the same data type.

## Remarks

Please refer to **F152_RMRD**

**s1_Control** stores the control data for the configuration of the master and slave units in the network. **n** words, beginning at the address in the CPU specified by **s2_Start**, are written to the intelligent unit of the slave unit beginning at the shared memory address number specified by **d_Start**.

Specifications of **s1_Control**:




(1) **s1** higher word
(2) Bit
(3) **s1** higher word
(4) Bank No. (16#00–16#FF if there is a bank to specify, otherwise 16#00)
(5) Slot No. (16#00–16#1F, FP3: 16#00–16#17)
(6) **s1** lower word
(7) **s1** lower word
(8) Master station No.(16#01–16#04)
(9) Slave station No. (16#01–16#20)

Tip

Intelligent unit with bank: FP3 expansion data memory unit

Order number: AFP32091 AFP32092

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the control data **s1_Control** exceeds the limit of specified range
- if no MEWNET-F master unit is found
- if the data read exceeds the area of **s2_Start**

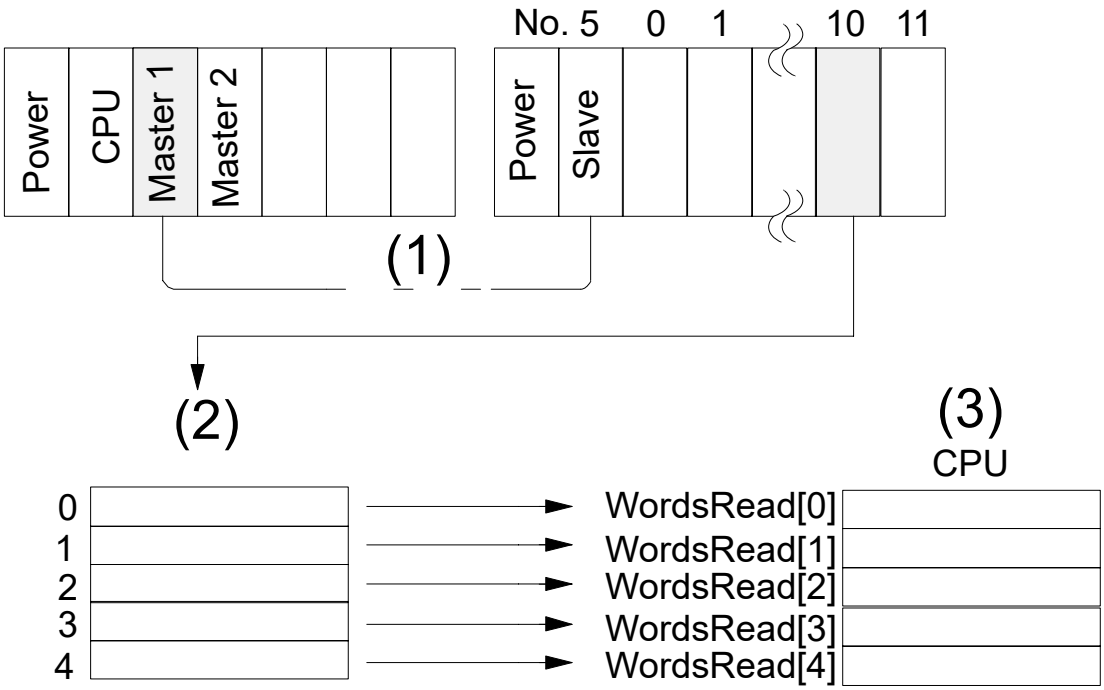
**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the control data **s1_Control** exceeds the limit of specified range
- if no MEWNET-F master unit is found
- if the data read exceeds the area of **s2_Start**

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.



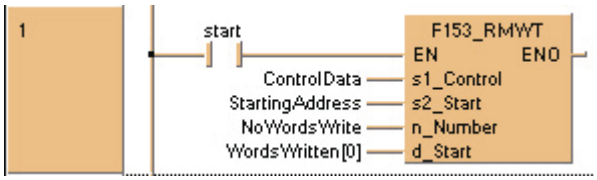

(1) Master station 1

(2) Intelligent unit (shared memory)

(3) Control data = 16#A0105

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | Start | BOOL | FALSE | |
| 1 | VAR | Control Data | DWORD | 16#2020A | No bank, slot no. 10, Master station 1, Slave station 5 |
| 2 | VAR | StartingAddress | WORD | 30 | |
| 3 | VAR | NoWordsWrite | INT | 10 | |
| 4 | VAR | WordsWritten | ARRAY [0..14] OF WORD | [15(0)] | |

## LD body



# 9 Communication instructions for program controlled mode

## 9.1 Sending data

Sending includes generating the data for the send buffer and sending it using the instructions **SendCharacters**, **SendCharactersAndClearString**, **SendData** or **F159_MTRN**.**SendCharacters** and **SendCharactersAndClearString** implicitly use **F159_MTRN**. The start and end codes specified in the system registers are automatically added to the data sent. The maximum volume of data that can be sent is 16384 bytes.

Procedure for sending data to external devices:

1. Set communication parameters
   Required settings: communication mode (program controlled), baud rate, communication format
2. Write to send buffer
   Not necessary when using **SendCharacters** or **SendCharactersAndClearString**.
3. Execute send command
   Use one of the following instructions:

| Instruction | Comment |
|---|---|
| **SendCharacters** | Easy to use, fits most applications, may require more data memory |
| **SendCharactersAndClearString** | Like **SendCharacters** but works without send buffer, may require less data memory |
| **F159_MTRN** | Original F instruction with complete set of parameters, additional transfer instruction required to write data to send buffer |

4. Optional: Evaluate "transmission done" flag
   Use one of the following methods:

| Method | Comment |
|---|---|
| **IsTransmissionDone** | Returns the value of the "transmission done" flag. It turns to TRUE when the specified number of bytes has been sent. |
| **sys_bIsComPort1TransmissionDone**<br>**sys_bIsComPort2TransmissionDone**<br>**sys_bIsToolPortTransmissionDone** | These system variables turn to TRUE when the specified number of bytes has been sent. |
| Input (X) flags X4 and X5 (MCU only) | These flags can be used to verify the end of transmission with a Multi-Communication Unit. |

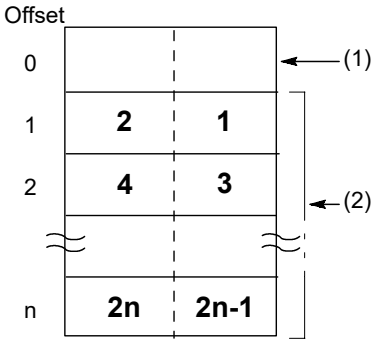
Note

- When the specified number of bytes has been sent, the "transmission done" flag turns to TRUE. New data may be sent or received. Any send instruction turns the "transmission done" flag to FALSE and no data can be received. Evaluation of the "transmission done" flag may be useful in cases where no response can be expected, e.g. with broadcast messages.
- Data cannot be sent unless the pin CS (Clear to Send) is on. When connecting to a three-wire port, short-circuit the RS and CS pins.

### 9.1.1 Writing to send buffer

The instructions **SendCharacters** and **SendCharactersAndClearString** automatically generate the data in the send buffer.

Send buffer layout




(1) Storage area for the number of bytes to be sent
(2) Storage area for the data to be sent

Bold numbers indicate the order of transmission. The storage area for the data to be sent starts with the second word of the send buffer (offset 1). Offset 0 contains the number of bytes to be sent and this register is internally handled by the PLC (users should use it read only). The maximum volume of data that can be sent is 2048 bytes.

If **F159_MTRN** is used for transmission, the data must be copied to the send buffer using a transfer instruction, e.g. **F10_BKMV**.

# IsTransmissionDone

Evaluate "transmission done" flag

This function returns the value of the "transmission done" flag. The "transmission done" flag is TRUE if the specified number of bytes has been sent from the assigned communication port of the PLC. New data may be sent or received. Any send instruction turns the "transmission done" flag to FALSE and no data can be received. Evaluation of the "transmission done" flag may be useful in cases where no response can be expected, e.g. with broadcast messages.


## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

Output

**IsDone** (BOOL)

TRUE, if the end code has been received. The end code is specified in the system registers.

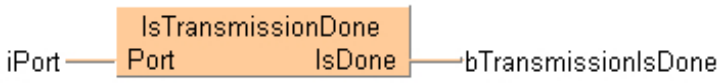
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iPort | INT | 0 |
| 1 | VAR | bTransmissionIsDone | BOOL | FALSE |

### LD body

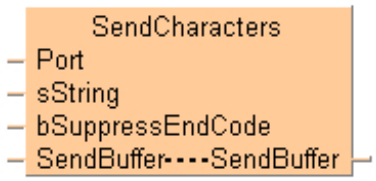

IsTransmissionDone
iPort
Port
IsDone
bTransmissionIsDone

# SendCharacters

Send characters via CPU or MCU port

This instruction writes data to the send buffer and executes **F159_MTRN** to send the data. The data to be sent is applied at **sString**. The send buffer is a VAR_IN_OUT variable applied at **SendBuffer**. If **bSuppressEndCode** is set to TRUE, the end code selected in the system registers is not appended to the send string.

In contrast to the instruction **SendCharactersAndClearString**, the string variable applied at **sString** remains unchanged.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**sString** (STRING)

Stores the send string

**bSuppressEndCode** (BOOL)

The end code selected in the system registers is not appended to the send string.

Input/output

**SendBuffer** (ANY)

Stores the send string temporarily

## Remarks

When the specified number of bytes has been sent, the "transmission done" flag turns to TRUE. New data may be sent or received. Any send instruction turns the "transmission done" flag to FALSE and no data can be received. Evaluation of the "transmission done" flag may be useful in cases where no response can be expected, e.g. with broadcast messages.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the MCU unit does not exist in the specified slot or zero bytes should be sent.
- if 16#8000 is specified in MEWTOCOL-COM Master/Slave mode

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the MCU unit does not exist in the specified slot or zero bytes should be sent.
- if 16#8000 is specified in MEWTOCOL-COM Master/Slave mode

## Example

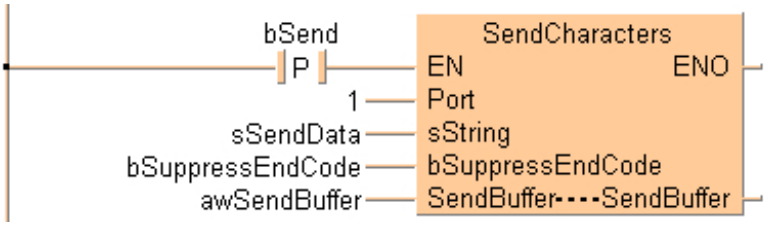
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bSend | BOOL | FALSE | activates function |
| 1 | VAR | sSendData | STRING[30] | 'ABCDEFGH' | up to 30 chars |
| 2 | VAR | awSendBuffer | ARRAY [0..15] OF WORD | [16(0)] | for 30 chars + 1 word |
| 3 | VAR | bSuppressEndCode | BOOL | FALSE | |

## LD body

If **bSend** changes from FALSE to TRUE, the instruction sends the characters of **sSendData** via the MCU port 1. The characters are copied to the array **awSendBuffer**. **awSendBuffer[0]** is reserved for the number of characters of the string to send.

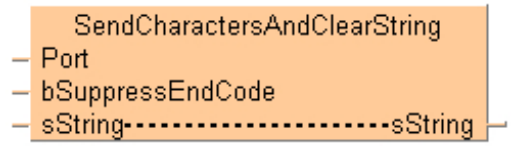


# SendCharactersAndClearString

Send characters and clear string

This instruction executes the send data instruction **F159_MTRN**. The data is contained in the send string applied at **sString**, a VAR_IN_OUT variable. If **bSuppressEndCode** is set to TRUE, the end code selected in the system registers is not appended to the send string.

In contrast to **SendCharacters**, no additional send buffer is used internally. The variable applied at **sString** is cleared after execution.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**bSuppressEndCode** (BOOL)

The end code selected in the system registers is not appended to the send string.

Input/output

**sString** (STRING)

Stores the send string, which is cleared after execution of the instruction

## Remarks

When the specified number of bytes has been sent, the "transmission done" flag turns to TRUE. New data may be sent or received. Any send instruction turns the "transmission done" flag to FALSE and no data can be received. Evaluation of the "transmission done" flag may be useful in cases where no response can be expected, e.g. with broadcast messages.

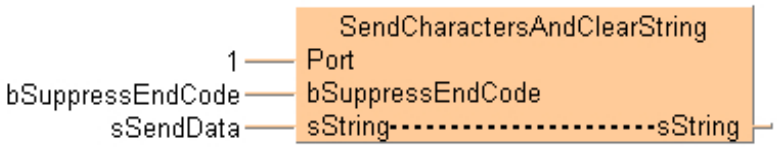
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | sSendData | STRING[30] | 'ABCDEFGH' | up to 30 chars |
| 1 | VAR | bSuppressEndCode | BOOL | FALSE | |

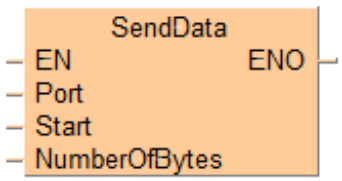
### LD body



# SendData

Send data via communication port or Ethernet user connection

This instruction sends data using a send buffer to external devices (computer, measuring instrument, bar code reader, etc.) connected to the specified communication port or via Ethernet user connection. If applied to the CPU's COM port, it also clears the receive buffer, resets the "reception done" flag and allows further reception of data.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**Start** (WORD, INT, UINT)

Send buffer

**NumberOfBytes** (WORD, INT, UINT)

Number of bytes to send:

- Negative value: The end code selected in the system registers is not appended to the send string.
- 0 (zero bytes): Prepare system to receive further data
- 16#8000: Toggle communication mode

## Remarks

- When the specified number of bytes has been sent, the "transmission done" flag turns to TRUE. New data may be sent or received. Any send instruction turns the "transmission done" flag to FALSE and no data can be received. Evaluation of the "transmission done" flag may be useful in cases where no response can be expected, e.g. with broadcast messages.
- **SendData** is encapsulated in the following instructions:

- **SendCharacters**
- **SendCharactersAndClearString**
- **ClearReceiveBuffer**
- **SetCommunicationMode**

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the number of bytes to be sent specified by **NumberOfBytes** is outside of the specified area.
- Flags only for the MCU:
  - if the MCU unit does not exist in the specified slot or zero bytes should be sent.
  - if 16#8000 is specified in MEWTOCOL-COM Master/Slave mode

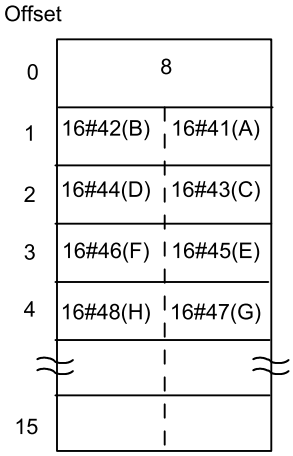
**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the number of bytes to be sent specified by **NumberOfBytes** is outside of the specified area.
- Flags only for the MCU:
  - if the MCU unit does not exist in the specified slot or zero bytes should be sent.
  - if 16#8000 is specified in MEWTOCOL-COM Master/Slave mode

## Example

In this example the characters of the string **sSendData** are transmitted. Define a send buffer for 30 bytes (ARRAY [0...15] OF WORD) and copy 8 characters of a string ("ABCDEFGH") into the buffer.

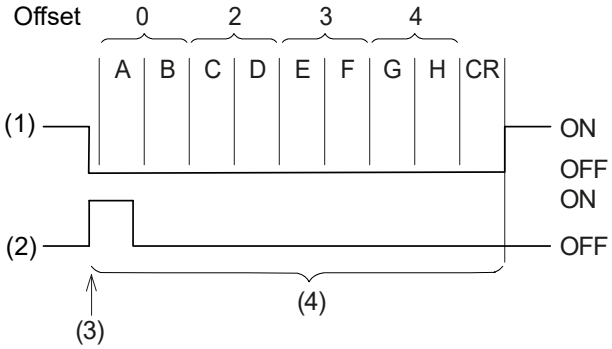
Send buffer layout:




The first word of the send buffer (offset 0) is reserved for the number of bytes to be sent. Therefore, copy the data into offset 1 (**SendBuffer[1]**).

When sending begins (the execution condition of the send instruction turns to TRUE), the value in offset 0 is set to 8. At the end of transmission, the value in offset 0 is automatically reset to 0. The data in offset 1 to offset 4 is sent in order from the lower byte.

Transmit the characters "ABCDEFGH" to an external device connected to COM port 1. For start code and end code the default settings "No-STX" and "CR" are selected.




(1) "Transmission done" flag
(2) Execution condition
(3) Execution of send instruction
(4) Transmission

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

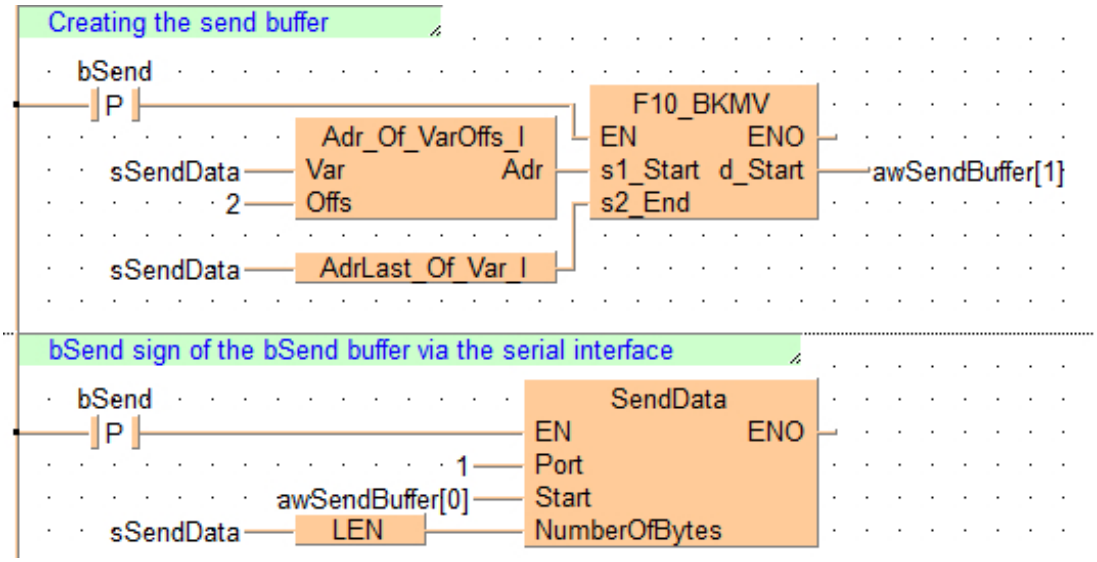
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bSend | BOOL | FALSE | activates function |
| 1 | VAR | sSendData | STRING[30] | 'ABCDEFGH' | up to 30 chars |
| 2 | VAR | awSendBuffer | ARRAY [0..15] OF WORD | [16(0)] | for 30 chars + 1 word |

## POU body

When **bSend** is TRUE, **F10_BKMV** copies the characters of the string at **sSendData** to the buffer **awSendBuffer** beginning at **awSendBuffer[1]**. The first two words of a string contain the string header information (maximum number of characters and current number of characters). The string header must not be copied into the buffer. Therefore, enter an offset of 2 to the starting address of the string before copying the data. Make sure the size of the send buffer is sufficient for all the data to be sent. Each element of the array at **SendBuffer** can contain two characters of the string at **SendString**. **SendBuffer[0]** is reserved for the total number of bytes to be sent.

**SendData** sends the data from the first element of the send buffer (**awSendBuffer[0])** as specified by **Start**. The length of the string to be sent (8 bytes) is set at **NumberOfBytes**. Use the function **LEN** to calculate the number of bytes. The data is output from COM port 1 as specified by **Port**.

## LD body



### 9.1.6 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

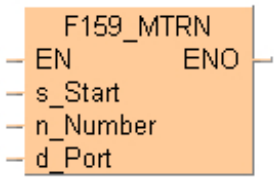
Advantages of IEC instructions

Advantages of FP instructions

# F159_MTRN

Send data via CPU or MCU port

This instructions sends data using a send buffer to external devices (computer, measuring instrument, bar code reader, etc.) connected to the specified communication port. If applied to the CPU's COM port, it also clears the receive buffer, resets the "reception done" flag and allows further reception of data.




Input

**s_Start** (WORD, INT, UINT)

Send buffer

**n_Number** (WORD, INT, UINT)

Number of bytes to send:

- Negative value: The end code selected in the system registers is not appended to the send string.
- 0 (zero bytes): Prepare system to receive further data
- 16#8000: Toggle communication mode between "Program controlled" and MEWTOCOL master/slave.

**d_Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

## Remarks

- When the specified number of bytes has been sent, the "transmission done" flag turns to TRUE. New data may be sent or received. Any send instruction turns the "transmission done" flag to FALSE and no data can be received. Evaluation of the "transmission done" flag may be useful in cases where no response can be expected, e.g. with broadcast messages.
- **F159_MTRN** is encapsulated in the following instructions:
  - **SendCharacters**
  - **SendCharactersAndClearString**

- **ClearReceiveBuffer**
- **SetCommunicationMode**

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the number of bytes to be sent specified by **n_Number** is outside of the specified area.
- Flags only for the MCU:
  - if the MCU unit does not exist in the specified slot or zero bytes should be sent.
  - if 16#8000 is specified in MEWTOCOL-COM Master/Slave mode

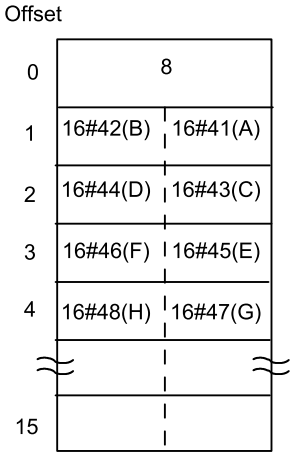
**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the number of bytes to be sent specified by **n_Number** is outside of the specified area.
- Flags only for the MCU:
  - if the MCU unit does not exist in the specified slot or zero bytes should be sent.
  - if 16#8000 is specified in MEWTOCOL-COM Master/Slave mode

## Example

In this example the characters of the string **sSendData** are transmitted. Define a send buffer for 30 bytes (ARRAY [0...15] OF WORD) and copy 8 characters of a string ("ABCDEFGH") into the buffer.

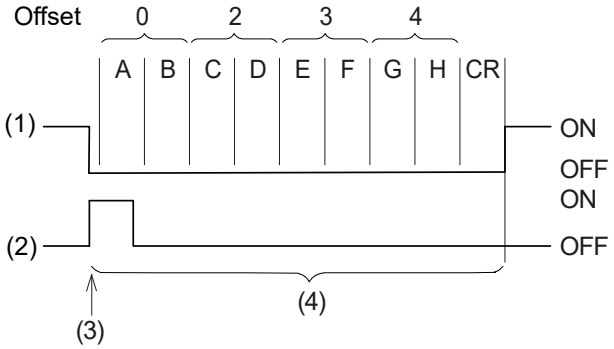
Send buffer layout:




The first word of the send buffer (offset 0) is reserved for the number of bytes to be sent. Therefore, copy the data into offset 1 (**SendBuffer[1]**).

When sending begins (the execution condition of the send instruction turns to TRUE), the value in offset 0 is set to 8. At the end of transmission, the value in offset 0 is automatically reset to 0. The data in offset 1 to offset 4 is sent in order from the lower byte.

Transmit the characters "ABCDEFGH" to an external device connected to COM port 1. For start code and end code the default settings "No-STX" and "CR" are selected.




1: "Transmission done" flag
2: Execution condition
3: Execution of send instruction
4: Transmission

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

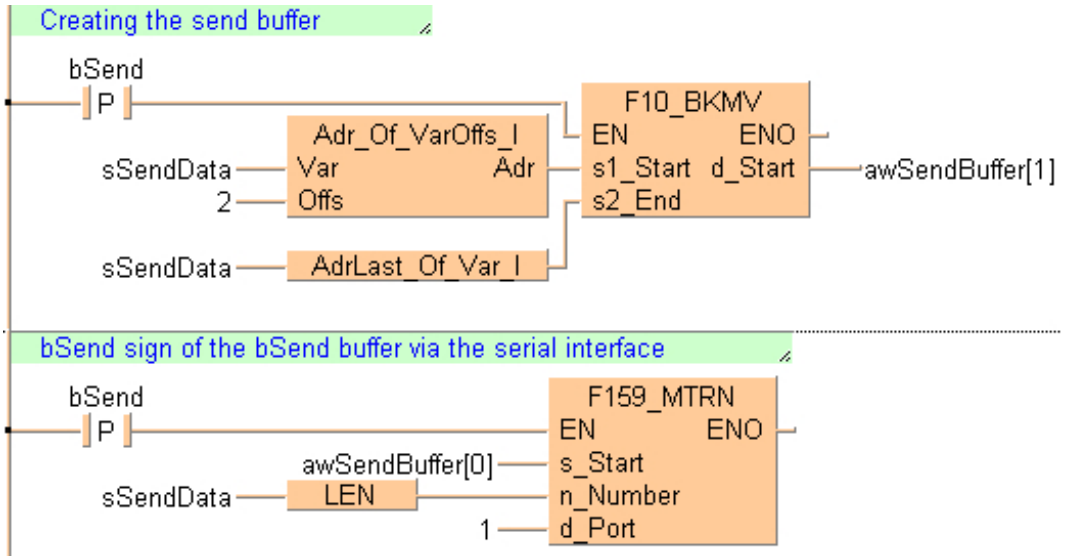
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bSend | BOOL | FALSE | activates function |
| 1 | VAR | sSendData | STRING[30] | 'ABCDEFGH' | up to 30 chars |
| 2 | VAR | awSendBuffer | ARRAY [0..15] OF WORD | [16(0)] | for 30 chars + 1 word |

### POU body

When **bSend** is TRUE, **F10_BKMV** copies the characters of the string at **sSendData** to the buffer **awSendBuffer** beginning at **awSendBuffer[1]**. The first two words of a string contain the string header information (maximum number of characters and current number of characters). The string header must not be copied into the buffer. Therefore, enter an offset of 2 to the starting address of the string before copying the data. Make sure the size of the send buffer is sufficient for all the data to be sent. Each element of the array at **SendBuffer** can contain two characters of the string at **SendString**. **SendBuffer[0]** is reserved for the total number of bytes to be sent.

**F159_MTRN** sends the data from the first element of the send buffer (**awSendBuffer[0]**) as specified by **s_Start**. The length of the string to be sent (8 bytes) is set at **n_Number**. Use the function **LEN** to calculate the number of bytes. The data is output from COM port 1 as specified by **d_Port**.

### LD body




## 9.2 Receiving data

Data can be received from an external device if the "reception done" flag is FALSE. (The "reception done" flag turns to FALSE after switching to “RUN mode” or if data is sent or the **ClearReceiveBuffer** is performed.) Data is automatically received in the receive buffer of the CPU or of the Multi-Communication Unit. For the CPU, the receive buffer must be defined in the system registers. After the end of reception has been verified, data can be copied into a specified target area of the CPU.

When for the specified time no further bytes are received or the instruction **ClearReceiveBuffer** is performed, the **IsDone** output turns to TRUE. Reception of any further data is prohibited. **F159_MTRN** or **ClearReceiveBuffer** turns the "IsDone" flag to FALSE.

Procedure for receiving data from external devices:

1. Set communication parameters and receive buffer

   Required settings: communication mode (program controlled), baud rate, communication format, receive buffer (CPU only)

2. Receive data

   Data is automatically received in the receive buffer.

3. Verify end of reception

   Use one of the following methods:

| Method | Comment |
|---|---|
| **IsReceptionDone** | Returns the value of the "reception done" flag. It is TRUE if the end code has been received. |
| **IsReceptionDoneByTimeOut** | Used to verify the end of reception by timeout, e.g. with binary data when no end code is expected.<br>Note<br>MCU: The reception timeout has to be set in the system register of the communication port. |
| **sys_bIsComPort1ReceptionDone**<br>**sys_bIsComPort2ReceptionDone**<br>**sys_bIsToolPortReceptionDone**<br>(CPU only) | These system variables turn to TRUE if the end code has been received. |
| Input (X) flags of MCU unit X0 and X2 (MCU only) | These flags can be used to verify the end of reception with a Multi-Communication Unit. |
| Direct evaluation of the receive buffer. | |

4. Process data in receive buffer

   Use one of the following instructions:

| Instruction | Comment |
|---|---|
| **ReceiveData** | Automatically copies data received by the CPU or Multi-Communication Unit into the specified variable. |
| **ReceiveCharacters** | Automatically copies characters received by the CPU or Multi-Communication Unit into a string variable. |
| **F10_BKMV** | Transfers data from the receive buffer to a target area. Not required with **ReceiveData** or **ReceiveCharacters**. |
| **F161_MRCV** | Copies data received by a Multi-Communication Unit to the CPU’s receive buffer. Not required with **ReceiveData** or **ReceiveCharacters**. |

5. Prepare CPU or MCU to receive next data

   Use one of the following instructions:

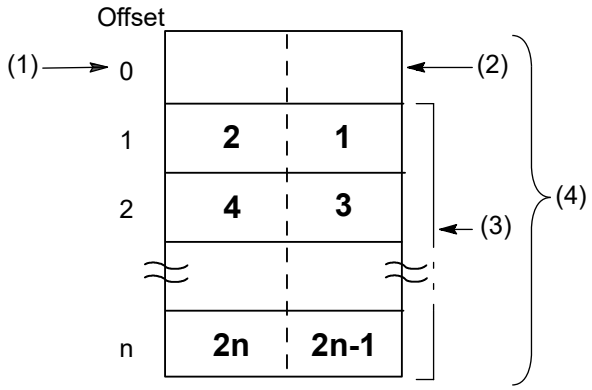
| Instruction | Comment |
|---|---|
| **ClearReceiveBuffer** | The receive buffer is automatically reset when sending the next data. To reset the receive buffer without sending any data use one of these instructions. |
| **SendData** | |

## 9.2.1 Setting receive buffer for CPU

For program controlled communication, a receive buffer must be specified in the DT memory area. The maximum area is up to 16384 words.Specify the following items:

1. Starting address
2. The capacity of the receive buffer (number of words)

Receive buffer layout




(1) Starting address
(2) Storage area for the number of bytes received
(3) Storage area for the data received
(4) Capacity

Bold numbers indicate the order of reception.Incoming data is stored in the receive buffer. Start and end codes are not stored in the receive buffer. The storage area for the data received starts with the second word of the receive buffer (offset 1). Offset 0 contains the number of bytes received. The initial value of offset 0 is 0.

1. Double-click “PLC” in the navigator.
2. Double-click “System registers”
3. Double-click “Serial ports”

   The communication ports occupy different bit positions of the same system register, so individual settings for each communication port are possible.To make settings for the TOOL port, select “System registers” > “Serial ports” > “TOOL”.The number of the system register for the respective settings may vary according to the PLC type used.

Note

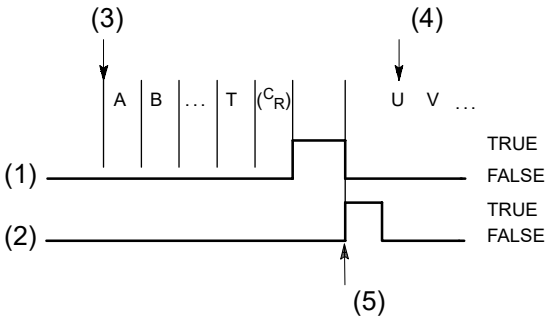
In order to use the data in the receive buffer, define a global variable having the same starting address and capacity. However FP7 and MCU expansion units do not have such a corresponding DT register address. To transfer data from the receive buffer into the PLC data register use **ReceiveData**

Related topics

- **ReceiveData**
- Communication parameter instructions
- Communication instructions for program controlled mode
- Communication parameter instructions

## Processing data in receive buffer and preparing CPU to receive further data

Receive a string of 8 bytes containing the characters "ABCDEFGH" via COM port 1. The characters are stored in ASCII HEX code without start and end codes.




(1) "Reception done" flag
(2) Execution condition
(3) Reception begins
(4) Reception continues
(5) Execution of **F159_MTRN (n_Number=0)**

Receive buffer layout:

| Offset | 8 | |
|---|---|---|
| 0 | | |
| 1 | 16#42(B) | 16#41(A) |
| 2 | 16#44(D) | 16#43(C) |
| 3 | 16#46(F) | 16#45(E) |
| 4 | 16#48(H) | 16#47(G) |

When reception begins, the value in offset 0 is 0. At the end of reception, the value in offset 0 is 8. The data in offset 1 to offset 4 is received in order from the lower byte.

### System register settings

| No | Item Name | Data | Dim... |
|---|---|---|---|
| 412 | COM port 1 communication mode | Program controlled... | |
| 410 | COM port 1 station number | 1 | |
| 415 | COM port 1 baud rate | 9600 | baud |
| 413 | COM port 1 sending data length | 8 bits | |
| 413 | COM port 1 sending parity check | With-Odd | |
| 413 | COM port 1 sending stop bit | 1 bit | |
| 413 | COM port 1 sending start code | No-STX | |
| 413 | COM port 1 sending end code/reception done condition | CR | |
| 416 | COM port 1 receive buffer starting address | 200 | |
| 417 | COM port 1 receive buffer capacity | 5 | |
| 412 | COM port 1 modem connection | Disable | |

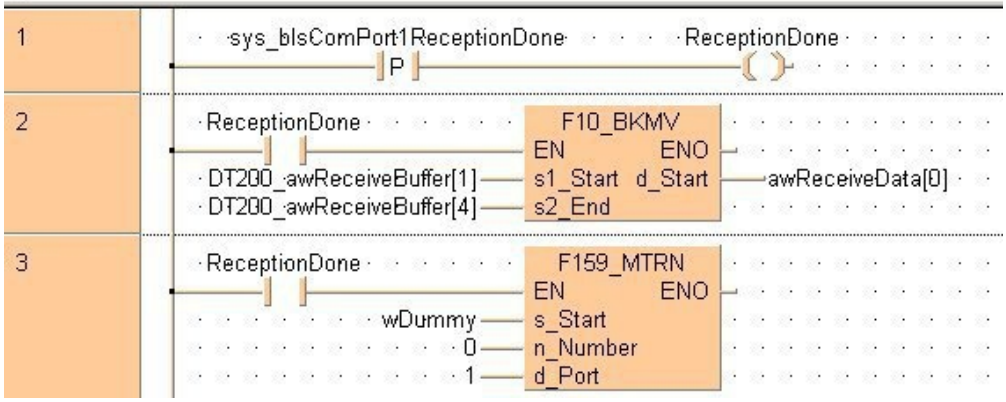
In order to use the data in the receive buffer, define a global variable having the same starting address and capacity. In this example, the starting address is DT200 (VAR_GLOBALDT200_awReceivedBuffer) and the receive buffer capacity is 5 (ARRAY [0..4] OF WORD).

### Global variables

| | Class | Identifier | FP A... | IEC Addr... | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | DT200_awReceiveBuffer | DT200 | %MW5.200 | ARRAY [0..4] OF WORD | [5(0)] |

### POU header and LD body

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | wDummy | WORD | 0 | |
| 1 | VAR | ReceptionDone | BOOL | FALSE | |
| 2 | VAR_EXTERNAL | DT200_awReceiveBuffer | ARRAY [0..4] OF WORD | [5(0)] | |
| 3 | VAR | awReceiveData | ARRAY[0..3] OF WORD | [4(0)] | |



Note

- The number of received bytes may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.
- The start code "STX" resets the receive buffer.

# ClearReceiveBuffer

Reset the receive buffer

This instruction clears the receive buffer at the communication port specified by **Port**.


## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

## Remarks

The receive buffer is automatically reset when a send instruction is executed. To reset the receive buffer without sending further data, execute this instruction. Alternatively, you can also use **F159_MTRN** with **n_Number** = 0. Resetting the receive buffer sets the number of bytes received in offset 0 to 0 and moves the write pointer back to offset 1. The next data will be stored starting at offset 1 and overwriting the existing data. The "reception done" flag turns to FALSE.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the communication port specified by **Port** does not exist.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the communication port specified by **Port** does not exist.

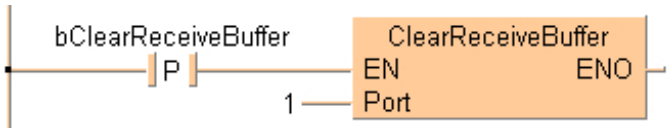
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bClearReceiveBuffer | BOOL | FALSE |

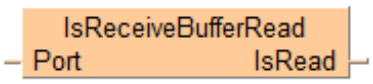
**LD body**



# IsReceiveBufferRead

Evaluate if the receive buffer is read

This instruction returns the value of the receive buffer read flag on a specified communication port.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

Output

**IsRead** (BOOL)

set to TRUE if the receive buffer has been read

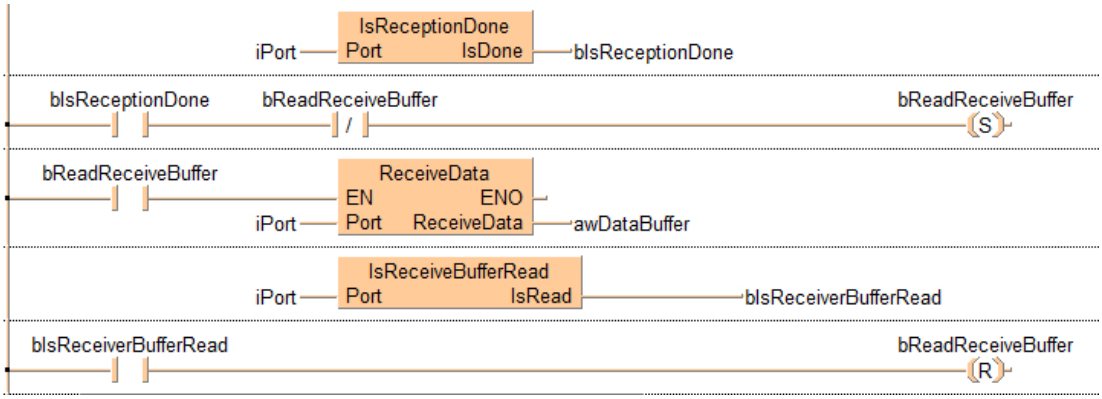
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iPort | INT | 0 |
| 1 | VAR | bIsReceptionDone | BOOL | FALSE |
| 2 | VAR | awDataBuffer | ARRAY [0..2] OF W... | [3(0)] |
| 3 | VAR | bIsReceiverBufferRead | BOOL | FALSE |
| 4 | VAR | bReadReceiveBuffer | BOOL | FALSE |

## LD body




Note

**IsReceiveBufferRead** is only active (=TRUE) directly after the **ReceiveData** command has been executed.

In the next PLC cycle **IsReceiveBufferRead** will not be active (=FALSE).

# IsReceptionDone

Evaluate "reception done" flag

This function returns the value of the "reception done" flag. It is TRUE if the end code has been received at the communication port assigned at **Port**.


## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

Output

**IsDone** (BOOL)

Set to TRUE if the end code has been received. The end code is specified in the corresponding system register under COM port settings.

## Remarks

Evaluation of the "reception done" flag

When for the specified time no further bytes are received or the instruction **ClearReceiveBuffer** is performed, the **IsDone** output turns to TRUE. Reception of any further data is prohibited. **F159_MTRN** or **ClearReceiveBuffer** turns the "IsDone" flag to FALSE. This function can be used for FP7 or MCU also for the timeout mode. The "reception done" flag can be evaluated using one of the following instructions or system variables:

- **IsReceptionDone**
- **IsReceptionDoneByTimeOut**
- **sys_bIsComPort1ReceptionDone**, **sys_bIsComPort2ReceptionDone**, **sys_bIsToolPortReceptionDone** (depending on the port)

The end of reception can also be determined by checking the contents of the receive buffer. The number of received bytes may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

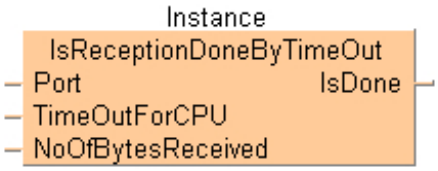
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iPort | INT | 0 |
| 1 | VAR | bIsDone | BOOL | FALSE |

### LD body

# IsReceptionDoneByTimeOut

Evaluate "reception done" condition by timeout

This function block evaluates a timeout condition to detect the end of reception in data streams not containing an end code, e.g. when transferring binary data.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**TimeOutForCPU** (TIME)

Time after which **IsDone** is set to TRUE if no further data is received.

**NoOfBytesReceived** (WORD, INT, UINT)

Applies to the starting address of the receive buffer. This address contains the number of bytes received. (CPU only)

Output

**IsDone** (BOOL)

Set to TRUE, if one or more bytes have been received and the number of bytes received has not changed within the time specified at **TimeOutForCPU** or in the "MCU Setting" dialog.

## Remarks

For the communication port of a CPU, the first word of the receive buffer must be applied at **NoOfBytesReceived** (number of bytes received). If the number of bytes received does not change within the time specified at **TimeOutForCPU**, **IsDone** turns to TRUE.

For a Multi-Communication Unit and for FP7, the "reception done" flag is evaluated. The timeout must be set in PROG mode using the “MCU Setting” dialog or in RUN mode using **F159_MWRT_PARA** (FP2 only).

Evaluation of the "reception done" flag

When for the specified time no further bytes are received or the instruction **ClearReceiveBuffer** is performed, the **IsDone** output turns to TRUE. Reception of any further data is prohibited. **F159_MTRN** or **ClearReceiveBuffer** turns the "IsDone" flag to FALSE.

The end of reception can also be determined by checking the contents of the receive buffer (e.g. the number of received bytes). The number of received bytes may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

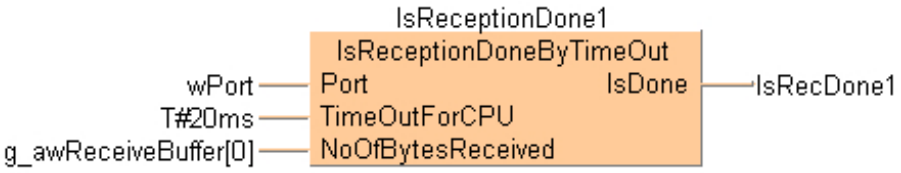
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | IsReceptionDone1 | IsReceptionDoneByTimeOut | |
| 1 | VAR_EXTERNAL | g_awReceiveBuffer | ARRAY [0..10] OF WORD | |
| 2 | VAR | IsRecDone1 | BOOL | FALSE |
| 3 | VAR | wPort | WORD | 0 |

### LD body



# IsRequestComPortReset

Evaluate the COM port reset requested

This instruction returns the value of the reset requested flag of a specified COM port.


## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

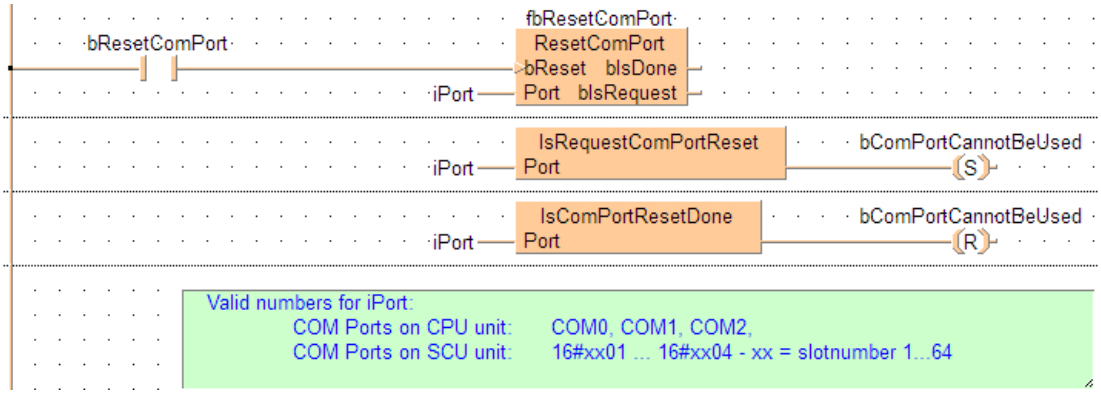
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | fbResetComPort | ResetComPort | |
| 1 | VAR | bResetComPort | BOOL | FALSE |
| 2 | VAR | iPort | INT | 0 |
| 3 | VAR | bComPortCannotBeUsed | BOOL | FALSE |
| 4 | VAR | bIsResetRequested | BOOL | FALSE |

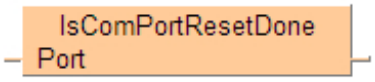
## LD body


fbResetComPort
bResetComPort
ResetComPort
bReset bIsDone
iPort
Port bIsRequest
IsRequestComPortReset
iPort
Port
bComPortCannotBeUsed
S
IsComPortResetDone
iPort
Port
bComPortCannotBeUsed
R
Valid numbers for iPort:
COM Ports on CPU unit: COM0, COM1, COM2,
COM Ports on SCU unit: 16#xx01 ... 16#xx04 - xx = slotnumber 1...64

# IsComPortResetDone

Evaluate if the COM port reset is done

This instruction returns the value of the reset done flag on a specified communication port.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

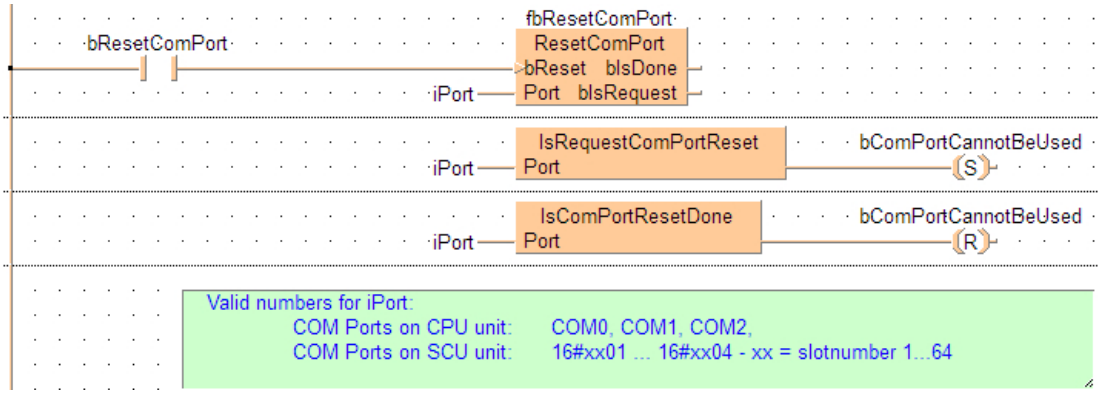
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | fbResetComPort | ResetComPort | |
| 1 | VAR | bResetComPort | BOOL | FALSE |
| 2 | VAR | iPort | INT | 0 |
| 3 | VAR | bComPortCannotBeUsed | BOOL | FALSE |
| 4 | VAR | bIsResetRequested | BOOL | FALSE |

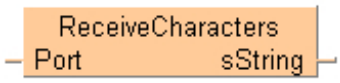
## LD body


fbResetComPort
bResetComPort
ResetComPort
bReset bIsDone
iPort
Port bIsRequest
IsRequestComPortReset
iPort
Port
bComPortCannotBeUsed
S
IsComPortResetDone
iPort
Port
bComPortCannotBeUsed
R
Valid numbers for iPort:
COM Ports on CPU unit: COM0, COM1, COM2,
COM Ports on SCU unit: 16#xx01 ... 16#xx04 - xx = slotnumber 1...64

# ReceiveCharacters

Receive characters from CPU or MCU port

This instruction copies the characters received at the port specified by **Port** into the variable applied at **sString**.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

Output

**sString** (STRING)

Stores the received characters

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

if the MCU unit does not exist at the slot no. specified by **Port**.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

if the MCU unit does not exist at the slot no. specified by **Port**.

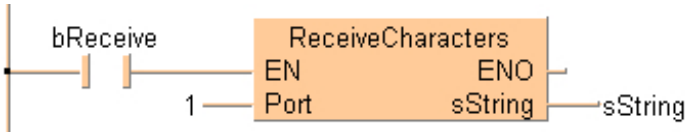
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bReceive | BOOL | FALSE | activates function |
| 1 | VAR | sString | String | '' | |

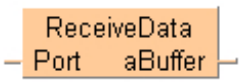
**LD body**



# ReceiveData

Receive data from CPU or MCU port

This instruction copies the data received at the port specified by **Port** into the variable applied at **aBuffer**.




## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

Output

**aBuffer** (ANY)

Stores the received data

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the MCU unit does not exist at the slot no. specified by '**Port**'.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the MCU unit does not exist at the slot no. specified by '**Port**'.

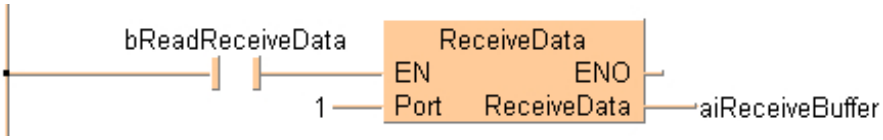
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bReadReceiveData | BOOL | FALSE |
| 1 | VAR | aiReceiveBuffer | ARRAY [0..10] OF INT | [11(0)] |

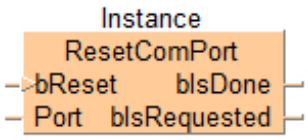
**LD body**



# ResetComPort

Reset a specified COM port

This function block sends a COM port reset request to a specified port number. The PLC has to be set to "program controlled" communication. Evaluate the output variables **bIsDone** and **bIsRequested** to check if the specified port is reset and can be used again.




## Parameters

Input

**bReset** (BOOL)

If set to TRUE, the COM port reset request is sent to the specified port number

**Port** (WORD, INT, UINT)

Port number

- COM0, COM1, COM2 on CPU
- 16#xx01–16#xx04 (xx = slot number 1–64) on SCU

Output

**bIsDone** (BOOL)

Set to TRUE, if the COM port is reset and can be used again.

**bIsRequested** (BOOL)

Set to TRUE, if the COM port reset is requested.

Corresponding system variables of outputs (set only, if the COM port has been reset by **bReset** of the same instance)

- Request to reset
  - COM0: **sys_bIsComPort0ResetRequested**
  - COM1: **sys_bIsComPort1ResetRequested**
  - COM2: **sys_bIsComPort2ResetRequested**
- Reset is done
  - COM0: **sys_bIsComPort0ResetDone**
  - COM1: **sys_bIsComPort1ResetDone**

– COM2: **sys_blsComPort2ResetDone**

## Example

### POU header

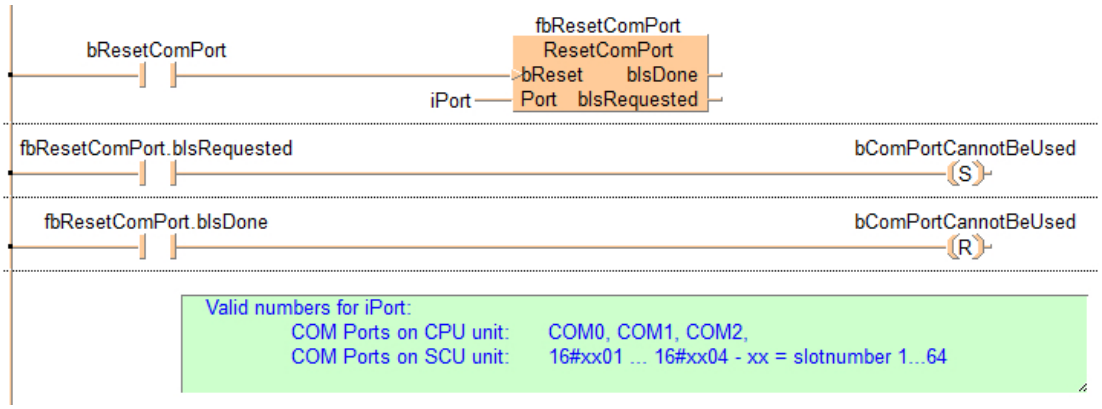
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | fbResetComPort | ResetComPort | |
| 1 | VAR | bResetComPort | BOOL | FALSE |
| 2 | VAR | iPort | INT | 0 |
| 3 | VAR | bComPortCannotBeUsed | BOOL | FALSE |

### POU body

When the variable **bResetComPort** is set to TRUE, the function block is carried out.

### LD body



### 9.2.11 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

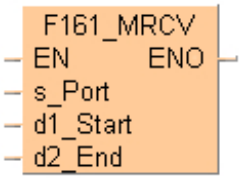
Advantages of IEC instructions

Advantages of FP instructions

# F161_MRCV

Read data from MCU port

Use this instruction to copy the data which the Multi-Communication Unit received from the external device to the CPU's receive buffer. The MCU's communication port is specified at **s_Port**. The receive buffer is defined by **d1_Start** and **d2_End**.




## Parameters

Input

**s_Port** (WORD, INT, UINT)

Specification of slot number (high byte) and port number (low byte) of the MCU to which the data is transmitted.
16#xx01: COM1 on MCU in slot 16#xx
16#xx02: COM2 on MCU in slot 16#xx

**d1_Start** (WORD, INT, UINT)

Starting address of the receive buffer

**d2_End** (WORD, INT, UINT)

Ending address of the receive buffer

## Remarks

- Do not execute **F161_MRCV** unless the end of reception has been verified by evaluating the "reception done" flag. Polling the data using **F161_MRCV** does not work correctly! The "reception done" flag can be evaluated using the **IsReceptionDone** and **IsReceptionDoneByTimeOut** functions or by evaluating the input (X) flags X0 and X2.
- The number of bytes received is stored in the initial address specified by **d1_Start** of the receive buffer. If the data received exceeds the ending address specified by **b2_End**, an operation error is detected. The data which has been received up to **d2_End** will be stored. **F161_MRCV** also clears the receive buffer, resets the "reception done flag" and allows further reception of data.
- **F161_MRCV** is supported by all PLCs: If suitable functions are used instead of flags, PLC-independent programs can be created which handle communication for CPU communication ports as well as for MCU ports. PLCs not using MCU ports simply do not

translate the **F161_MRCV** instruction. It is recommended to use the functions **ReadData** or **ReadCharacters** for a program that is easier to read.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the MCU unit does not exist in the specified slot or zero bytes should be sent.
- if the specified communication port does not exist

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the MCU unit does not exist in the specified slot or zero bytes should be sent.
- if the specified communication port does not exist

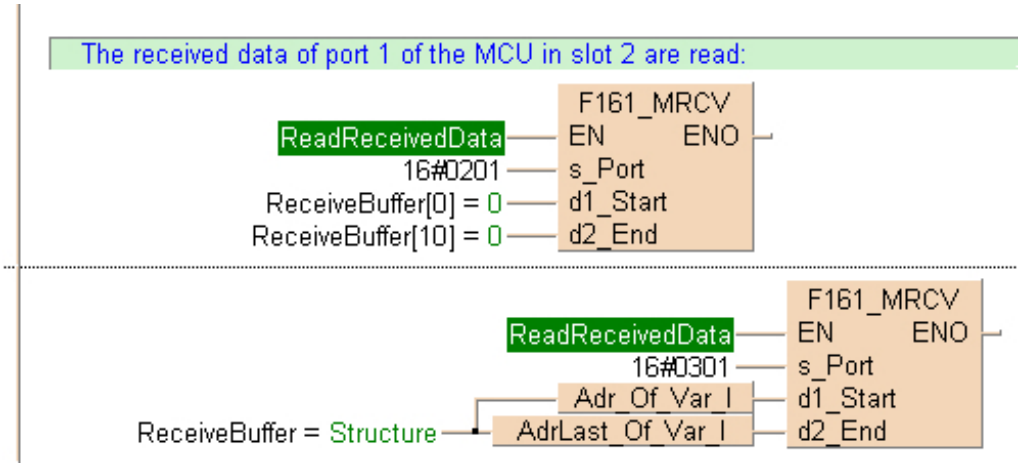
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | ReadReceivedData | BOOL | FALSE |
| 1 | VAR | ReceiveBuffer | ARRAY [0..10] OF INT | [11(0)] |

## LD body



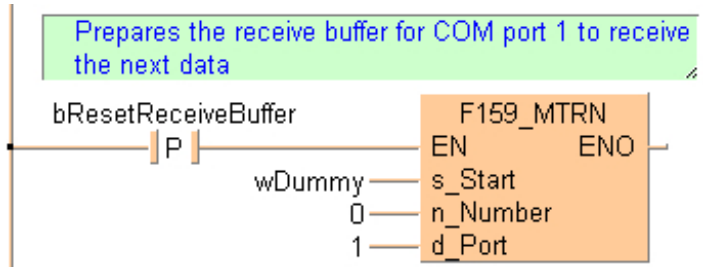
### Example

#### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bResetReceiveBuffer | BOOL | FALSE |
| 1 | VAR | wDummy | WORD | 0 |

#### LD body




## 9.3 Communication errors

If an error occurs during communication, the communication error flag turns to TRUE.

Select one of the following methods to evaluate the communication error flag:

| Method | CPU | MCU |
|---|---|---|
| **IsCommunicationError** | ● | |
| **sys_bIsComPort1CommunicationError**<br>**sys_bIsComPort2CommunicationError**<br>**sys_bIsToolPortCommunicationError** | ● | |
| Input (x) flags X6 and X7 | | ● |

Note

If the communication error flag turns to TRUE during reception, reception continues. Execute a send instruction to turn the error flag to FALSE and to move the write pointer back to offset 1.

# IsCommunicationError

Evaluate communication error flag

This instruction returns the value of the communication error flag. The communication error flag is TRUE if an error has occurred during serial communication at the communication port specified by **Port**.


## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

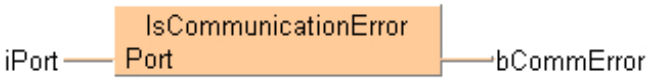
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iPort | INT | 0 |
| 1 | VAR | bCommError | BOOL | FALSE |

### LD body



# 10 Communication parameter instructions

## 10.1 Setting communication parameters

The communication mode and other communication parameters are set in the PLC's system registers.

Each communication mode requires different settings.

| Parameter | Program controlled communication | Master/slave communication (MEWTOCOL-COM/Modbus-RTU) | “PLC link” |
|---|---|---|---|
| Station number | – | ● | ● |
| Baud rate[1)] | ● | ● | Fixed |
| Data length, stop bit, parity, end code, start code [1)] | ● | ●[2)] | Fixed |
| Receive buffer (CPU only) | ● | – | – |
| Link area | – | – | ● |

[1)] The setting must match the external device connected to the communication port.

[2)] The end code setting must always be "CR", and the start code setting must be "No STX".

Use the programming tool or DIP switches on the PLC to make your settings in PROG mode. Or use F instructions to change parameters in RUN mode.

Select the appropriate setting method for your CPU or Multi-Communication Unit:

| **Parameter** | **PROG mode** | **RUN mode** | **CPU** | **MCU** |
|---|---|---|---|---|
| Communication mode | System registers | – | ● | ● |
| | – | **SetCommunicationMode** | ● | ● |
| | – | **F159_MTRN** (n_Number=16#8000) | ● | ● |
| Other parameters (e.g. baud rate, station number, start and end code, receive buffer) | System registers or DIP switches | – | ● | |
| | – | **SYS1**, **SYS2** | ● | |
| | MCU dialog or DIP switches | – | | ● |
| | – | **F159_MWRT_PARA** | | ● |

Related topics

MCU parameter settings

Communication parameter instructions (page 528)

DIP switch settings of Multi-Communication Unit (page 536)

**SetCommunicationMode** (page 532)

**F159_MTRN (n_Number=16#8000)** (page 567)

**SYS1** Communication condition setting for the COM ports of the CPU (page 1861)

**SYS2** (page 1878)

**F159_MWRT_PARA** (page 569)

Tip

FP-X0, Multi-Communication Unit: For PDF files, please refer to Panasonic Download Center

### 10.1.1 Setting communication parameters in system registers

1. Double-click “PLC” in the navigator.
2. Double-click “System registers”
3. Double-click “Serial ports”
   The communication ports occupy different bit positions of the same system register, so individual settings for each communication port are possible. To make settings for the TOOL port, select “System registers” > “Serial ports” > “TOOL”. The number of the system register for the respective settings may vary according to the PLC type used.
4. Make settings for the communication mode, communication format, baud rate, station number, and receive buffer if necessary.

Tip

FP-X0, Multi-Communication Unit: For PDF files, please refer to Panasonic Download Center

### 10.1.2 Communication ports

To specify the communication port, you can use the following system variables or constants for the input 'Port':

| COM ports on the CPU | Ethernet port on the CPU<br>(FP7 E types) | MCU (FP2/2SH)<br>SCU (FP7) |
|---|---|---|
| **SYS_COM0_PORT**<br>**SYS_COM1_PORT**<br>**SYS_COM2_PORT**<br>**SYS_COM3_PORT**<br>(FPXH)<br>**SYS_TOOL_PORT** | **SYS_ETHERNET_USER_CONNECTION_2**...<br>**SYS_ETHERNET_USER_CONNECTION_216** | 16#xx01 (COM1)<br>16#xx02 (COM2)<br>xx = slot number (hexadecimal) of the MCU/SCU, e.g.:<br>• 16#0001: COM1 on the CPU<br>(slot 0 SCU only)<br>• 16#0A02: COM2 in slot 10<br>(MCU/SCU)<br>• 16#1401: COM1 in slot 20<br>(MCU/SCU) |

Tip

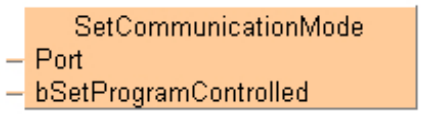
FP-X0, Multi-Communication Unit: Please refer to the corresponding manual that you can find in the download center in the Internet.

# SetCommunicationMode

Set communication mode

This instruction sets the communication mode to “Program controlled” or to “MEWTOCOL-COM slave”/“MEWTOCOL-COM master/slave”, depending on the value at **bSetProgramControlled**:

- TRUE: “Program controlled”
- FALSE: “MEWTOCOL-COM slave”/“MEWTOCOL-COM master/slave”




## Parameters

Input

**Port** (INT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**bSetProgramControlled** (BOOL)

Sets the communication mode:

- TRUE: “Program controlled”
- FALSE: “MEWTOCOL-COM slave”/“MEWTOCOL-COM master/slave”

## Remarks

- When the power is turned on, the communication mode selected in the system registers is set.
- It is not possible to change to Modbus RTU mode or PLC Link mode during RUN mode.

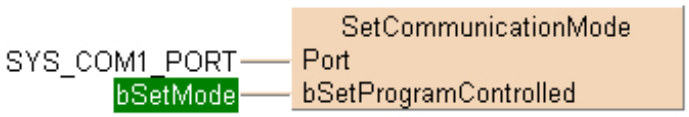
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bSetMode | BOOL | FALSE | If TRUE, communication mode is set to 'Program controlled' |

## LD body



# SetProgramControlled

Set communication mode to “Program controlled”

This instruction sets the communication mode to “Program controlled”


## Parameters

Input

**Port** (INT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

## Remarks

- When the power is turned on, the communication mode selected in the system registers is set.
- It is not possible to change to Modbus RTU mode or PLC Link mode during RUN mode.

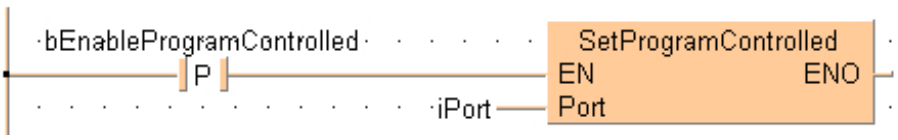
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Klasse | Bezeichner | Typ | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnableProgramControlled | BOOL | FALSE |
| 1 | VAR | iPort | INT | 0 |

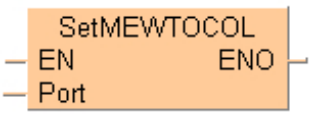
### LD body



# SetMEWTOCOL

Set communication mode

This instruction sets the communication mode to “MEWTOCOL-COM slave”/“MEWTOCOL-COM master/slave”.




## Parameters

Input

**Port** (INT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

## Remarks

- When the power is turned on, the communication mode selected in the system registers is set.
- It is not possible to change to Modbus RTU mode or PLC Link mode during RUN mode.

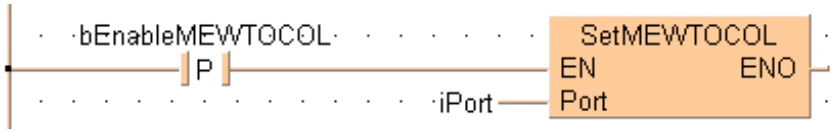
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Klasse | Bezeichner | Typ | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnableMEWTOCOL | BOOL | FALSE |
| 1 | VAR | iPort | INT | 0 |

### LD body



## 10.1.6 DIP switch settings of Multi-Communication Unit

The DIP switches are located at the back of the unit and can be used to set the communication mode and baud rate. The DIP switches must also be set as indicated below if you want to use the MCU Configurator or **F159_MWRT_PARA**.

Note

- To enable any settings made with the MCU Configurator, set DIP switches 3 and 4 of the MCU to ON for COM port 1. For COM port 2 set switches 7 and 8 to ON.
- The factory setting for all DIP switches is ON.

### DIP switch settings

| | **Port** | **COM 1** | | | | **COM 2** | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Switch No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Communication mode | (Not used) | OFF | OFF | – | – | OFF | OFF | – | – |
| | PLC link | ON | OFF | – | – | ON | OFF | – | – |
| | Program controlled communication | OFF | ON | – | – | OFF | ON | – | – |
| | MEWTOCOL-COM Slave | ON | ON | – | – | ON | ON | – | – |
| Baud rate | 115200bit/s | – | – | OFF | OFF | – | – | OFF | OFF |
| | 19200bit/s | – | – | ON | OFF | – | – | ON | OFF |
| | 9600bit/s | – | – | OFF | ON | – | – | OFF | ON |
| | Use MCU Configurator or **F159_MWRT_PARA** | OFF or ON[1)] | OFF or ON[1)] | ON | ON | – | – | ON | ON |

[1)] Depending on the selected communication mode

### 10.1.7 FP instructions

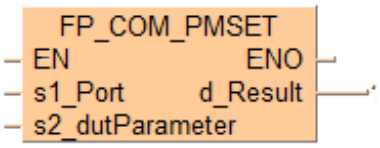
Tip

Advantages of FP instructions

# FP_COM_PMSET

Set communication parameters in SCU

This FP instruction sends changes to communication parameters to the unit. Changes are specified by **s2_dutParameter** for the communication port specified by **s1_Port**.




## Parameters

Input

**s1_Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

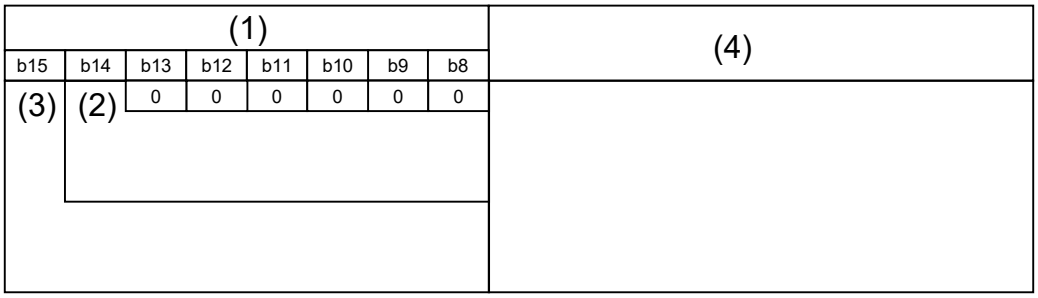
**s2_dutParameter** (**FP_COM_PARAMETER_DUT**)

Parameters to be set for SCU port

Output

**d_Result** (WORD, INT, UINT)

Starting address of the memory area in the master unit that stores the processing result (1 word)
Content of the processing result **d_Result**




(1) Higher byte

(2) Execution result flag
FALSE: Normal completion
TRUE: Abnormal completion

(3) Process-in-progress flag
FALSE: Process is completed
TRUE: Process is in progress

(4) Lower byte

0: Normal completion

1: The specified communication port is invalid

2: Setting error

3: Mode change error

4: The specified communication port is occupied

5: Inconsistency of parameters to be changed (at the start/end of the setting process, parameters specified by operands are inconsistent)

## Remarks

- Before you execute the instruction, make sure that the bit 15 (process in-progress flag) of the processing result storage area specified by **dResult** is FALSE.
- If parameter change is carried out for a COM port where sending/receiving is in progress, the sending/receiving process is cancelled and parameters are changed. At this time, received data are lost. The sending process is suspended.
- This instruction cannot be called by user-defined functions. Use only in programs and function blocks.
- Change the communication parameters for the COM port of the CPU unit, using a user program.
- While the requested change is being processed, Bit 15 of the processing result storage area **d_Result** turns to TRUE. When the process is completed, it turns to FALSE.
- The processing result is stored in the area specified by **d_Result**. If an error occurs, the execution result flag (bit 14) is turned to TRUE. The error code is stored in lower bytes of **d_Result**.
- By reading setting parameters using the **FP_COM_GET_PARAMETER** instruction, and setting parameters to be changed using the **FP_COM_PMSET** instruction, the settings can be simplified.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

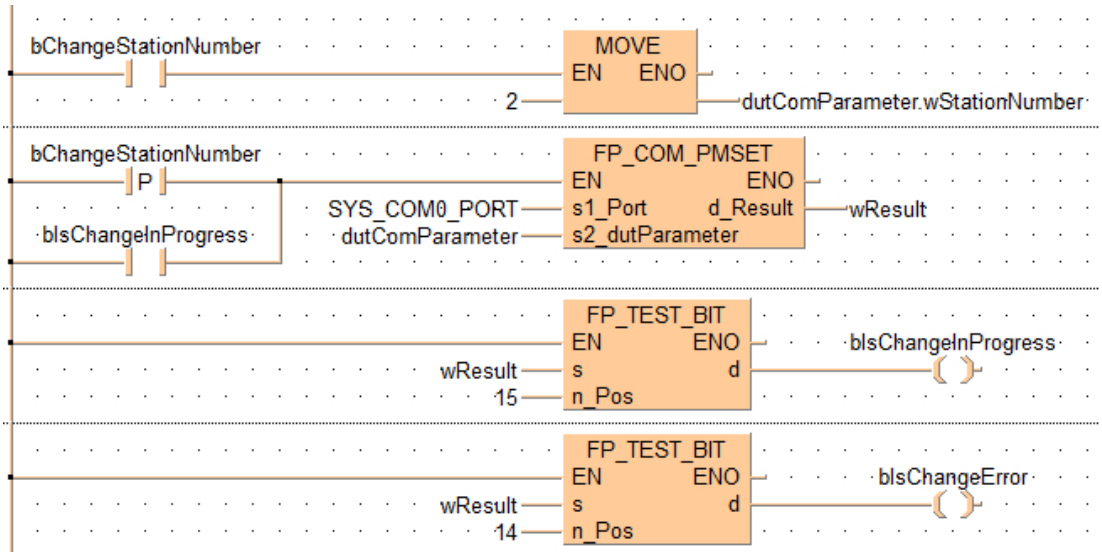
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | dutComParameter | FP_COM_PARAMETER_DUT | |
| 1 | VAR | bChangeStationNumber | BOOL | FALSE |
| 2 | VAR | wResult | WORD | 0 |
| 3 | VAR | bIsChangeInProgress | BOOL | FALSE |
| 4 | VAR | bIsChangeError | BOOL | FALSE |

## LD body



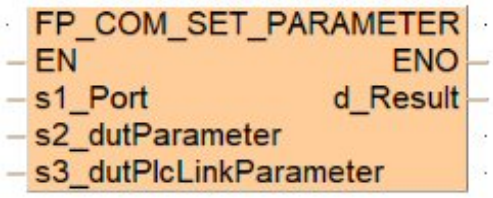
# FP_COM_SET_PARAMETER

Set communication parameters for CPU/SCU/MCU port

This FP instruction sends changes to communication parameters to the unit. Changes are specified by **s2_dutParameter** for the communication port specified by **s1_Port**. If the communication mode is set to “PLC link”, the PLC link parameters can be specified by **s3_dutPlcLinkParameter**.

Note

When using this instruction, please make sure that the communication mode of the appropriate port is set to “PLC link”.




## Parameters

Input

**s1_Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

**s2_dutParameter** (**FP_COM_PARAMETER_DUT**)

Parameters to be set for CPU/SCU/MCU port

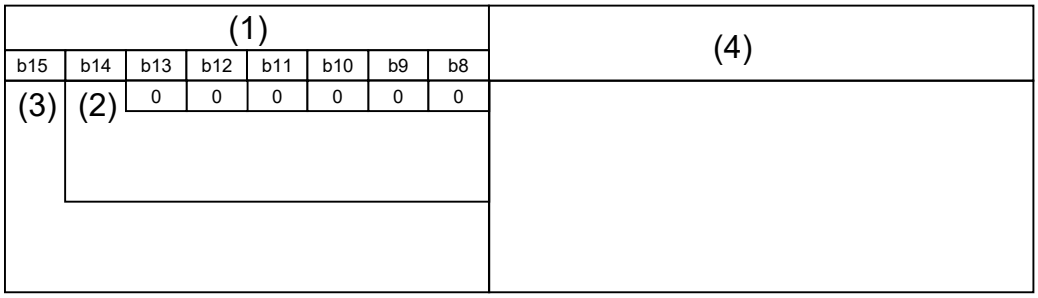
**s3_dutPlcLinkParameter** (**FP_COM_PLCLINK_PARAMETER_DUT**)

Parameters to be set for “PLC link”

Output

**d_Result** (WORD, INT, UINT)

Starting address of the memory area in the master unit that stores the processing result (1 word)
Content of the processing result **d_Result**




(1) Higher byte

(2) Execution result flag
FALSE: Normal completion
TRUE: Abnormal completion

(3) Process-in-progress flag
FALSE: Process is completed
TRUE: Process is in progress

(4) Lower byte (set only when processing is done)

0: Normal completion

1: The specified communication port is invalid

2: Setting error

3: Mode change error

4: The specified communication port is occupied

5: Inconsistency of parameters to be changed (at the start/end of the setting process, parameters specified by operands are inconsistent)

7: Unit number setting on the front panel of the multi-wire link unit (when the rotary switch is not set to 0)

## Remarks

- Before you execute the instruction, make sure that the bit 15 (process in-progress flag) of the processing result storage area specified by **dResult** is FALSE.
- If parameter change is carried out for a COM port where sending/receiving is in progress, the sending/receiving process is cancelled and parameters are changed. At this time, received data are lost. The sending process is suspended.
- While the requested change is being processed, Bit 15 of the processing result storage area **d_Result** turns to TRUE. When the process is completed, it turns to FALSE.
- The processing result is stored in the area specified by **d_Result**. If an error occurs, the execution result flag (bit 14) is turned to TRUE. The error code is stored in lower bytes of **d_Result**.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the communication mode of the appropriate port is not set to “PLC link”

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

- if the communication mode of the appropriate port is not set to “PLC link”

## Example

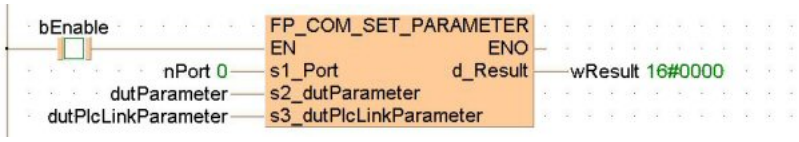
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | nPort | INT | |
| 2 | VAR | dutParameter | FP_COM_PARAMETER_DUT | |
| 3 | VAR | dutPlcLinkParameter | FP_COM_PLCLINK_PARAMETER_DUT | |
| 4 | VAR | wResult | WORD | 0 |
| 5 | VAR | bEnable | BOOL | FALSE |

### LD body

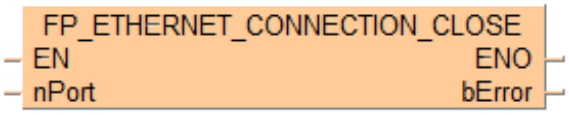
When the variable **bEnable** is set to TRUE, the function is executed.



# FP_ETHERNET_CONNECTION_CLOSE

Close an Ethernet connection

This FP instruction closes an open Ethernet connection on the port specified by **nPort**. Please ensure the suitable parameters are also set in the project navigator under “System registers” > “Ethernet” > “User connections”.




## Parameters

Input

**nPort** (WORD, INT, UINT)

Ethernet port on CPU (FP7 E types):
**SYS_ETHERNET_USER_CONNECTION_1**–**SYS_ETHERNET_USER_CONNECTION_216**

Output

**bError** (BOOL)

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Example

## POU header

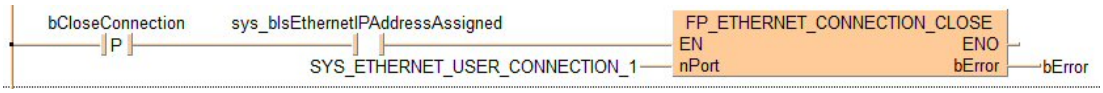
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetConnection | BOOL | FALSE |
| 1 | VAR | bOpenConnection | BOOL | FALSE |
| 2 | VAR | bCloseConnection | BOOL | FALSE |

## POU body

When the variable **bCloseConnection** changes from FALSE to TRUE and the system variable **sys_blsEthernetIPAddressAssigned** is set to TRUE, the function is carried out.

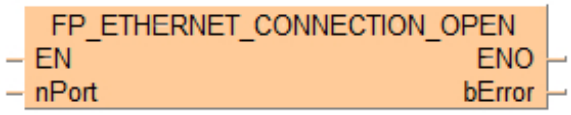
## LD body



# FP_ETHERNET_CONNECTION_OPEN

Open an Ethernet connection

This FP instruction opens an Ethernet connection on the port specified by **nPort**. Please ensure the suitable parameters are also set in the project navigator under “System registers” > “Ethernet” > “User connections”.




## Parameters

Input

**nPort** (WORD, INT, UINT)

Ethernet port on CPU (FP7 E types):
**SYS_ETHERNET_USER_CONNECTION_1**–**SYS_ETHERNET_USER_CONNECTION_216**

Output

**bError** (BOOL)

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Example

## POU header

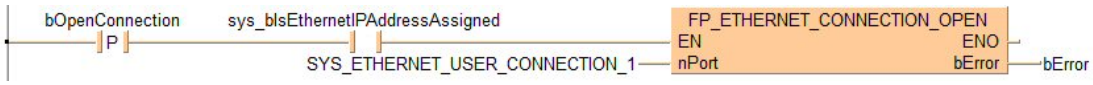
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetConnection | BOOL | FALSE |
| 1 | VAR | bOpenConnection | BOOL | FALSE |
| 2 | VAR | bCloseConnection | BOOL | FALSE |

## POU body

When the variable **bOpenConnection** changes from FALSE to TRUE and the system variable **sys_bIsEthernetIPAddressAssigned** is set to TRUE, the function is carried out.

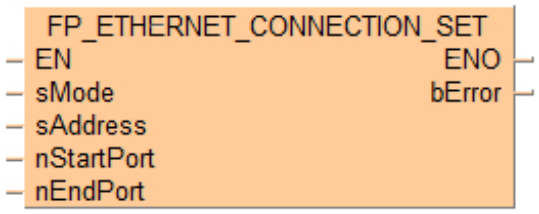
## LD body



# FP_ETHERNET_CONNECTION_SET

Set an Ethernet connection

This FP instruction sets up an Ethernet connection with settings specified by **sMode** and **sAddress** in a range of user connections specified by **nStartPort** and **nEndPort**. Please ensure the suitable parameters are also set in the project navigator under “System registers” > “Ethernet” > “User connections”.




## Parameters

Input

**sMode** (STRING)

Set the communication mode and connection parameters.

- Communication mode (required parameter)

  Specify parameters for the communication mode.

  Keywords:

  - `'MEWCOM'`: MEWTOCOL-COM (default)
  - `'MEW7COM'`: MEWTOCOL7-COM
  - `'MODBUS'`: MODBUS-TCP
  - `'MEWDAT'`: MEWTOCOL-DAT
  - `'GP'`: Program controlled [General purpose] (only available for 16 user connections)
  - `'GP_LARGE'`: Program controlled [General purpose] with large capacity reception (only available for the first user connection and only on PLC types CPS4* and CPS3* version 4.32 or higher)

- Option setting (required parameter)

  Some communication modes have an additional protocol option that can be activated. Specify whether or not to use the option for the selected communication mode.

  - `'OPTAV'`: The option available for the selected communication mode is activated
  - `'OPTNAV'`: Either there is no option available for the selected communication mode or the available option is not activated (default)

| Communication mode | 'OPTAV' | 'OPTNAV' |
|---|---|---|
| MEWTOCOL-COM | Connect with FP2 ET-LAN | Do not connect |
| MEWTOCOL7-COM | Not available | Not available |
| MODBUS-TCP | Not available | Not available |
| MEWTOCOL-DAT | Connect with FP2 ET-LAN | Do not connect |
| Program controlled [General purpose] | Do not append a special header | Append a special header |
| Program controlled [General purpose] with large capacity reception | Do not append a special header | Not available. If this keyword is specified, an operation error occurs. |

- Specify the type of Ethernet connection to be opened (required parameter).
  - 'CL': Client connection (default)

    For client connections, the destination unit IP address is incremented by one for each user connection from the user connection specified by **nStartPort** to the user connection specified by **nEndPort**.
  - 'SV': Server connection

    For server connections, the master unit port number is incremented by one for each user connection from the user connection specified by **nStartPort** to the user connection specified by **nEndPort**.
- Specify the method for opening the Ethernet connection (required parameter).
  - 'AUTO': Open the connection automatically (default)
  - 'MANU': Do not open the connection automatically (instead, use the instruction **FP_ETHERNET_CONNECTION_OPEN**)
- Specify the type of Ethernet communication (required parameter).
  - 'TCP': TCP/IP setting (default)
  - 'UDP': UDP/IP setting (not available for the communication mode GP_LARGE)

  **Notes:**
  - If you have specified UDP as the communication type, there is a difference between using FP instructions and making configuration settings in the project navigator. There is no parameter "open type" available in the project navigator. However, "open type" must be specified in the FP instructions. Use the keywords 'SV' for a slave connection and 'CL' for a master connection.
  - Do not specify UDP for the communication mode GP_LARGE because this will lead to an operation error.
- Reset to default values

  'INITIAL': Resets all settings to the default settings

Examples:

1. Set MEWTOCOL-COM as the communication mode, use the option to connect with FP2 ET-LAN, a client connection should be established and opened automatically, and communication type is TCP/IP: 'MEWCOM,OPTAV,CL,AUTO,TCP'

2. Set MODBUS-TCP as the communication mode, option is not available, a server connection should be established but not opened automatically, and the communication type is UDP/IP: `'MODBUS,OPTNAV,SV,MANU,UDP'`
3. Set the communication mode to program controlled, append a special header, a server connection should be established and opened automatically, and the communication type is UDP/IP: `'GP,OPTNAV,SV,AUTO,UDP'`
4. Set the communication mode to program controlled with large capacity reception, do not append a special header, a server connection should be established but not opened automatically, and the communication type is TCP/IP: `'GP_LARGE,OPTAV,SV,MANU,TCP'`
5. When you want to reset the configuration to the default settings so that communication mode is MEWTOCOL-COM, no connection to FP2 ET-LAN, a client connection should be established and opened automatically, and communication type is TCP/IP: `'INITIAL'`

**sAddress** (STRING)

Set the address parameters. The settings differ depending on whether you are specifying client or server connections.

For client connections, you need to specify the IP addresses and the port numbers of the destination units as well as the disconnect time for unused connections.

For server connections, you need to specify the port numbers of the master unit and the disconnect time for unused connections.

- Specifying a client connection
  - Destination unit IP address (required parameter)

    Specify the destination unit IP address for the first user connection in the range of user connections specified by **nStartPort** and **nEndPort**. The IP address for the other user connections is set by incrementing the last block of the IP address by one. The setting for the destination port number and the disconnect time is the same for all user connections and will not be incremented.

    For an IP address, specify the keyword `IPv4` or `IPv6` at the beginning.

    — Syntax for IPv4: e.g. `'IPv4=111.122.133.144'`

    — Syntax for IPv6: e.g. `'IPv6=1111:122:2:1555:0:0:1888'`

    Please note that for IPv4 addresses there are range restrictions. For IPv4, 000.000.000.000 (0.0.0.0) cannot be specified. When an invalid IP address is specified with an instruction, there will be no operation error but the output **bError** will be set to TRUE. The system variable **sys_iEthernetConnectionErrorCode** is set to "1: Incorrect IP address is specified".
  - Destination unit port number (required parameter)

    Keyword: `'PORT'`

    Syntax:`'PORT=xxxxx'`

    Values: 1–65535
  - Disconnect time for unused connections (required parameter)

    Keyword: `'DISCONT'`

    Syntax: `'DISCONT=xxxx'`

Values: 0–4294967295ms in steps of 10ms. 0: The connection is not automatically disconnected.

– Receive buffer starting address (optional parameter)

This parameter is only available for FP0H C32ET/EP and FP-XH C40ET, C60ET, FP-XH C60ETF when the communication mode is set to program controlled [General purpose].

Keyword: `'BUFTOP'`

Syntax: `'BUFTOP=xxxx'`

Values: DT0 to last possible DT

– Size of receive buffer (optional parameter)

This parameter is only available for FP0H C32ET/EP and FP-XH C40ET, C60ET, FP-XH C60ETF when the communication mode is set to program controlled [General purpose].

Keyword: `'BUFSIZE'`

Syntax: `'BUFSIZE=xxxx'`

Values: 0–2048

Examples:

1. Set destination unit address to 192.244.2.10 and destination unit port number to 9000, do not disconnect automatically even if connection is unused: `'IPv4=192.244.2.10,PORT=9000,DISCONT=0'`
2. Set destination unit address to 1222::1555:0:0:1999 and destination unit port number to 10000, disconnect automatically after connection has not been used for 30 seconds: `'IPv6=1222::1555:0:0:1999,PORT=10000,DISCONT=30000'`
3. Set destination unit address to 192.168.1.5 and destination unit port number to 4000, disconnect automatically after connection has not been used for 50 seconds, set starting address of receive buffer to DT4 and receive buffer size to 256: `'IPv4=192.168.1.5, PORT=4000, DISCONT=100, BUFTOP=DT4, BUFSIZE=256'`

• Specifying a server connection

– Master unit port number (required parameter)

Keyword: `'PORT'`

Syntax: `'PORT=xxxxx'`

Values: 1–65535

– Disconnect time for unused connections (required parameter)

Keyword: `'DISCONT'`

Syntax: `'DISCONT=xxxx'`

Values: 0–4294967295ms in steps of 10ms. 0: The connection is not automatically disconnected.

Examples:

1. Set master unit port number to 9000, do not disconnect automatically even if connection is unused: `'PORT=9000,DISCONT=0'`
2. Set master unit port number to 10000, disconnect automatically after connection has not been used for 30 seconds: `'PORT=10000,DISCONT=30000'`

3. Set master unit port number to 11111, disconnect automatically after connection has not been used for 70 milliseconds: `'PORT=11111,DISCONT=70'`

**nStartPort** (WORD, INT, UINT)

Set the number of the first user connection
**SYS_ETHERNET_USER_CONNECTION_1**–**SYS_ETHERNET_USER_CONNECTION_216**

**nEndPort** (WORD, INT, UINT)

Set the number of the last user connection
**SYS_ETHERNET_USER_CONNECTION_1**–**SYS_ETHERNET_USER_CONNECTION_216**

Output

**bError** (turns to TRUE and remains TRUE) (BOOL)

- if an IP address is invalid
- if the instruction is executed while the IP address is not established

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- Do not change the order of keywords. Specify the keywords and their setting parameters in the order they are listed here.
- Use a comma "," to separate a keyword and its setting from the next keyword.
- Be sure that **nStartPort** ≤ **nEndPort**
- Be sure to specify **nStartPort** and **nEndPort** so that all destination unit IP addresses (client connection) or all master unit port numbers (server connection) are inside the permissible range.
- The maximum number for Ethernet user connections is 216.
- If you execute the instruction when a user connection is open or a user connection is set to open automatically, an operation error occurs. However, when multiple user connections are set with the instruction, the setting changes for the user connections **before** the user connection in which an operation error occurs will be done. The setting changes for the user connections **after** the user connection in which an operation error occurs will not be done.
- If you use this instruction for multiple client connections some of which have already been configured, an operation error occurs because user connections which have been configured for one server are bundled together and will all be opened and closed at the same time.
- This instruction is not available in interrupt programs.
- Both upper and lower case characters can be used. "Abcd", "ABCD" and "abcd" are synonymous.
- Before you execute the instruction, make sure that **sys_bIsEthernetInitializing** is FALSE. **sys_bIsEthernetInitializing** turns to TRUE when the instruction is executed.

When you execute the instruction while **sys_bIsEthernetInitializing** is TRUE, an error occurs.

- Before you execute the instruction, make sure that **sys_bIsEthernetIPAddressAssigned** is TRUE. When you execute the instruction while **sys_bIsEthernetIPAddressAssigned** is FALSE, **bError** turns to TRUE and the instruction is terminated without being executed.
- When an incorrect IP address is specified, **bError** is set and no operation is executed.
- When this instruction has been executed successfully, the system variables **sys_bIsCarry** and **sys_iEthernetConnectionErrorCode** are reset.
- This instruction does not overwrite the Ethernet configuration data stored in the PLC permanently. When the PLC has been switched off and on again, the Ethernet configuration data stored in the PLC are used again.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if **nStartPort** > **nEndPort**
- if the user connection number is outside the permissible range
- if a value specified for a parameter is outside the permissible range.
- if the same keyword is specified more than once
- if a port number is invalid
- if the number of characters for string data exceeds 256.
- if there is an open connection
- if there is a connection set to open automatically
- if one or more user connections in the range specified by **nStartPort** and **nEndPort** belong to an already existing server configuration.
- if GP_LARGE is specified as the communication mode and OPTNAV is specified as the protocol option
- if GP_LARGE is specified as the communication mode and UDP is specified as the Ethernet communication type
- if GP_LARGE is specified as the communication mode and **nStartPort** or **nEndPort** is not 1
- if the instruction is executed in an interrupt program

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if **nStartPort** > **nEndPort**
- if the user connection number is outside the permissible range
- if a value specified for a parameter is outside the permissible range.
- if the same keyword is specified more than once

- if a port number is invalid
- if the number of characters for string data exceeds 256.
- if there is an open connection
- if there is a connection set to open automatically
- if one or more user connections in the range specified by **nStartPort** and **nEndPort** belong to an already existing server configuration.
- if GP_LARGE is specified as the communication mode and OPTNAV is specified as the protocol option
- if GP_LARGE is specified as the communication mode and UDP is specified as the Ethernet communication type
- if GP_LARGE is specified as the communication mode and **nStartPort** or **nEndPort** is not 1
- if the instruction is executed in an interrupt program

## Example

### POU header

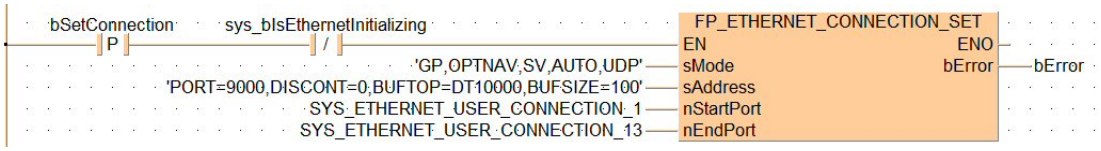
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetConnection | BOOL | FALSE |
| 1 | VAR | bOpenConnection | BOOL | FALSE |
| 2 | VAR | bCloseConnection | BOOL | FALSE |

### POU body

When the variable **bSetConnection** changes from FALSE to TRUE and the system variable **sys_bIsEthernetInitializing** is not TRUE, the function is carried out.

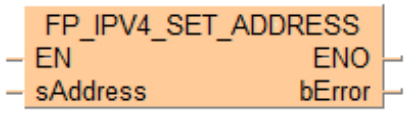
### LD body



# FP_IPV4_SET_ADDRESS

Set IP address using IPv4 protocol

This FP instruction sets up a connection to the IP address specified by **sAddress** via internet protocol V4.




## Parameters

Input

**sAddress** (STRING)

- IPv4 address (required parameter)
  Keyword: `IP`
- Netmask (optional parameter)
  Keyword: `MASK`
- Gateway (optional parameter)
  Keyword: `GWIP`
  Specify "0" when the default gateway is not to be used.

Examples:

1. Set the IP address to 192.168.1.5, use netmask 255.255.255.0 and the gateway 192.168.1.1: `'IP=192.168.1.5,MASK=255.255.255.0,GWIP=192.168.1.1'`
2. Set the IP address to 192.168.1.5, do not use a netmask, use the gateway 192.168.1.1: `'IP=192.168.1.5,,GWIP=192.168.1.1'`

Output

**bError** (BOOL)

- if a timeout of the connection is exceeded
- if an IP address is invalid

When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Remarks

- It is recommended to execute this instruction only once at the startup of PLC. Do not execute it repeatedly.
- It takes three seconds or longer to complete initialization.

- The number of characters for string data must not exceed 256.
- This instruction is not available in interrupt programs.
- Ensure the string at **sAddress** does not contain any blanks. Use comma as separator character only.
- IPv4 address is mandatory. Netmask and gateway can be omitted.
- A part of parameters can be omitted. The settings are not changed when parameters are omitted partially.
- Upper and lower case characters can be used for specifying keywords.
- Do not change the order of keywords. Specify the keywords and their setting parameters in the order they are listed here.
- Before you execute the instruction, make sure that **sys_blsEthernetInitializing** is FALSE. **sys_blsEthernetInitializing** turns to TRUE when the instruction is executed. When you execute the instruction while **sys_blsEthernetInitializing** is TRUE, an error occurs.
- Before you execute the instruction, check if the system variable **sys_blsEthernetIPAddressAssigned** is TRUE. If it is FALSE when executing the instruction, an error occurs.
- The instruction can only be executed when **sys_blsEthernetCableNotConnected** is FALSE.
- When an error occurs, check the system variable **sys_iEthernetConnectionErrorCode** for the error code number.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.
- if the same keyword is specified more than once
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.
- if the same keyword is specified more than once
- if the instruction is executed in an interrupt program
- if the number of characters for string data exceeds 256.

**sys_blsCarry** (turns to TRUE for one scan)

- if the instruction is executed with an incorrect IP address, netmask, gateway or a combination of incorrect IP addresses **sys_iEthernetConnectionErrorCode** is set to the corresponding error code (1–4).

- if the instruction is executed while the Ethernet cable is disconnected. **sys_iEthernetConnectionErrorCode** is set to "10: Ethernet cable disconnected".
- if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### POU header

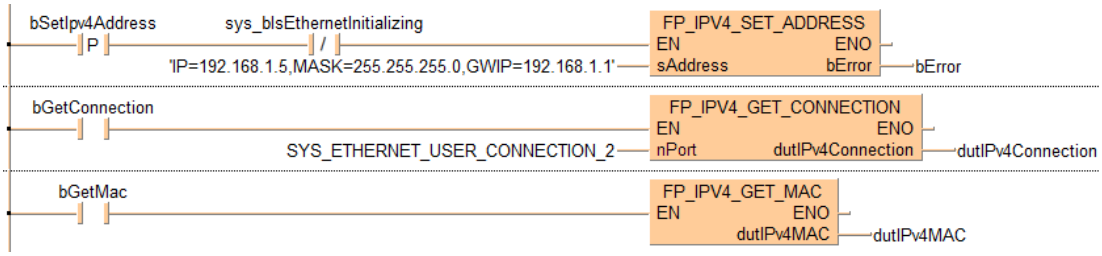
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetIpv4Address | BOOL | FALSE |
| 1 | VAR | bGetConnection | BOOL | FALSE |
| 2 | VAR | bGetMac | BOOL | FALSE |
| 3 | VAR | dutIPv4Connection | FP_IPv4_CONNECTION_DUT | |
| 4 | VAR | dutIPv4MAC | FP_IPv4_MAC_DUT | |

### POU body

This example sets the Ethernet connection parameters specified by **s_Address**. Please ensure the suitable parameters are also set in the project navigator under “System registers” > “Ethernet” > “IP addresses”. When **bGetConnection** is set to TRUE, the connection parameters are entered into the DUT **FP_IPv4_CONNECTION_DUT**. When **bGetMac** is set to TRUE, the parameters of the MAC address are entered into the DUT **FP_IPv4_MAC_DUT**.

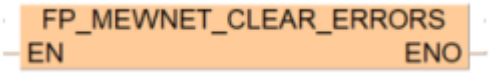
### LD body



# FP_MEWNET_CLEAR_ERRORS

Clear errors in the FP7 multi-wire link unit

This FP instruction clears errors in the FP7 multi-wire link unit. It resets the values of the following system variables:




| System variable | Description |
|---|---|
| **sys_blsMultiWireUnit1Error** | Error in FP7 multi-wire link unit 1 |
| **sys_blsMultiWireUnit2Error** | Error in FP7 multi-wire link unit 2 |
| **sys_blsMultiWireUnit3Error** | Error in FP7 multi-wire link unit 3 |
| **sys_blsMultiWireUnit4Error** | Error in FP7 multi-wire link unit 4 |
| **sys_blsMultiWireUnit5Error** | Error in FP7 multi-wire link unit 5 |
| **sys_blsMultiWireUnit6Error** | Error in FP7 multi-wire link unit 6 |
| **sys_wMultiWireUnit1ErrorCode** | FP7 multi-wire link unit 1 error (higher byte: error code, lower byte: unit number) |
| **sys_wMultiWireUnit2ErrorCode** | FP7 multi-wire link unit 2 error (higher byte: error code, lower byte: unit number) |
| **sys_wMultiWireUnit3ErrorCode** | FP7 multi-wire link unit 3 error (higher byte: error code, lower byte: unit number) |
| **sys_wMultiWireUnit4ErrorCode** | FP7 multi-wire link unit 4 error (higher byte: error code, lower byte: unit number) |
| **sys_wMultiWireUnit5ErrorCode** | FP7 multi-wire link unit 5 error (higher byte: error code, lower byte: unit number) |
| **sys_wMultiWireUnit6ErrorCode** | FP7 multi-wire link unit 6 error (higher byte: error code, lower byte: unit number) |

## Example

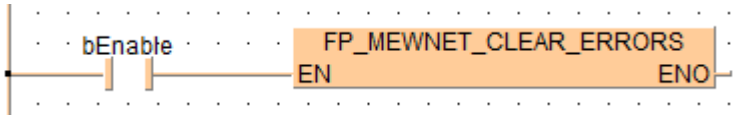
## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |

### LD body

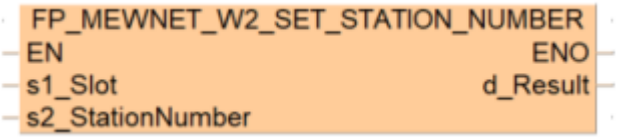
When the variable **bEnable** is set to TRUE, the function is executed.



# FP_MEWNET_W2_SET_STATION_NUMBER

Set MEWNET-W2 station number

This FP instruction sets the MEWNET-W2 station number for the MEWNET-W2 unit in the slot number specified by **s1_Slot**.




Input

**s1_Slot** (ANY16)

Slot number of expansion unit

**s2_StationNumber** (ANY16)

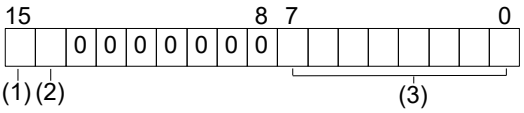
Station number (setting range: 1–64)
The station number can be changed only when the rotary switch on the front panel of the unit is set to 0.

Output

**d_Result** ((ANY16))

Starting address of the memory area in the master unit that stores the processing result (1 word)

Content of the processing result **d_Result**




(1) Bit 15: Process-in-progress flag
FALSE: Process is completed
TRUE: Process is in progress

(2) Bit 14: Execution result flag
FALSE: Normal completion
TRUE: Abnormal completion

(3) Lower byte: Execution result code

0: Normal completion

1: The specified communication port is invalid

2: Setting error

3: Mode change error

4: The specified communication port is occupied

5: Inconsistency of parameters to be changed (at the start/end of the setting process, parameters specified by operands are inconsistent)

7: Unit number setting on the front panel of the multi-wire link unit (when the rotary switch is not set to 0)

## Remarks

- Before you execute the instruction, make sure that the bit 15 (process in-progress flag) of the processing result storage area specified by **dResult** is FALSE.
- The processing result is stored in the area specified by **d_Result**.If an error occurs, the execution result flag (bit 14) is turned to TRUE. The error code is stored in lower bytes of **d_Result**.

## Example

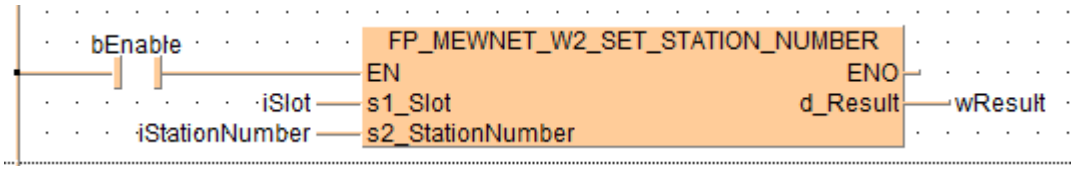
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | iSlot | INT | 0 |
| 3 | VAR | wResult | WORD | 0 |
| 4 | VAR | iStationNumber | INT | 0 |

### LD body

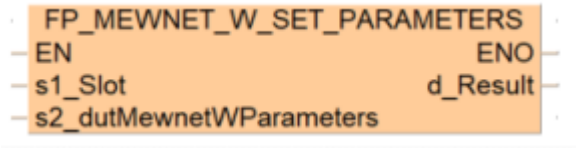
When the variable **bEnable** is set to TRUE, the function is executed.



# FP_MEWNET_W_SET_PARAMETERS

Set MEWNET-W parameters

This FP instruction sets the MEWNET-W parameters for the MEWNET-W unit in the slot specified by **s1_Slot**.




Input

**s1_Slot** (ANY16)

Slot number of expansion unit

**s2_dutMewnetWParameters** (**FP_MEWNET_W_PARAMETERS_DUT**)

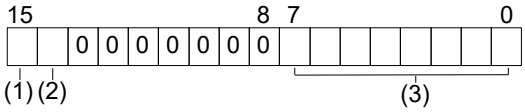
Parameters to be set in the MEWNET-W unit

Output

**d_Result** (ANY16)

Starting address of the memory area in the master unit that stores the processing result (1 word)

Content of the processing result **d_Result**




(1) Bit 15: Process-in-progress flag
FALSE: Process is completed
TRUE: Process is in progress

(2) Bit 14: Execution result flag
FALSE: Normal completion
TRUE: Abnormal completion

(3) Lower byte: Execution result code

0: Normal completion

1: The specified communication port is invalid

2: Setting error

3: Mode change error

4: The specified communication port is occupied

5: Inconsistency of parameters to be changed (at the start/end of the setting process, parameters specified by operands are inconsistent)

7: Unit number setting on the front panel of the multi-wire link unit (when the rotary switch is not set to 0)

### Remarks

- Before you execute the instruction, make sure that the bit 15 (process in-progress flag) of the processing result storage area specified by **dResult** is FALSE.
- The processing result is stored in the area specified by **d_Result**.If an error occurs, the execution result flag (bit 14) is turned to TRUE. The error code is stored in lower bytes of **d_Result**.

### Example

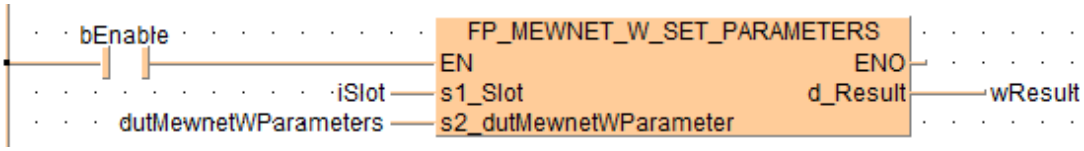
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | iSlot | INT | 0 |
| 3 | VAR | dutMewnetWParameters | FP_MEWNET_W_PARAMETERS_DUT | |
| 4 | VAR | wResult | WORD | 0 |

### LD body

When the variable **bEnable** is set to TRUE, the function is executed.

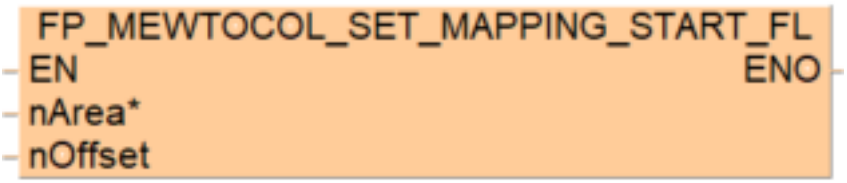


# FP_MEWTOCOL_SET_MAPPING_START_FL

Set a replacement register for FL addresses

This instruction sets a replacement register for FL addresses when the FP7 communicates with other FP-series controllers via MEWTOCOL.

This FP instruction sets a replacement DT/LD register in the FP7 CPU if an FL register is specified by the partner FP-series PLC for MEWTOCOL-COM or MEWTOCOL-DAT communication. The instruction can be used for an FP7 with built-in SCU, built-in ET-LAN, or connected serial communication unit and is compatible with FP2SH.




Input

**nArea*** (WORD, INT, UINT)

Memory area for the register replacing the FL register. (**SYS_MEMORY_AREA_DT** or **SYS_MEMORY_AREA_LD**)

**nOffset** (ANY32)

Start address offset for the replacement of FL0, e.g. 100123 for DT100123.

## Remarks

- Set this instruction to be executed only one time after switching to “RUN mode”.
- This instruction cannot be executed while the built-in Ethernet is being initialized. Before you execute the instruction, make sure that **sys_bIsEthernetIPAddressAssigned** is set to FALSE.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the offset specified at **nOffset** exceeds the limit of the memory area

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the offset specified at **nOffset** exceeds the limit of the memory area

## Example

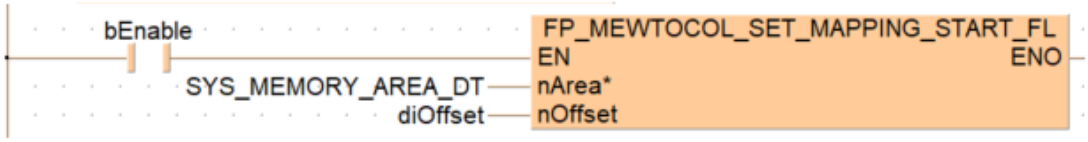
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | diOffset | DINT | 123456 |

### LD body

When the variable **bEnable** is set to TRUE, the function is executed.



### 10.1.8 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

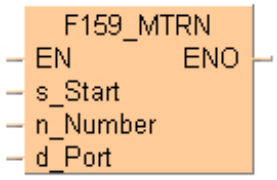
Advantages of IEC instructions

Advantages of FP instructions

# F159_MTRN (n_Number=16#8000)

Toggle communication mode

The communication mode of the CPU's communication ports can be changed during RUN mode. You can toggle between program controlled mode and MEWTOCOL-COM mode by executing **F159_MTRN** and setting the variable **n_Number** (the number of bytes to be sent) to 16#8000.




## Parameters

Input

**s_Start** (WORD, INT, UINT)

Send buffer

**n_Number** (WORD, INT, UINT)

Number of bytes to send:

- Negative value: The end code selected in the system registers is not appended to the send string.
- 0 (zero bytes): Prepare system to receive further data
- 16#8000: Toggle communication mode

**d_Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

## Remarks

- When the power is turned on, the communication mode selected in the system registers is set.
- It is not possible to change to Modbus RTU mode or PLC Link mode during RUN mode.

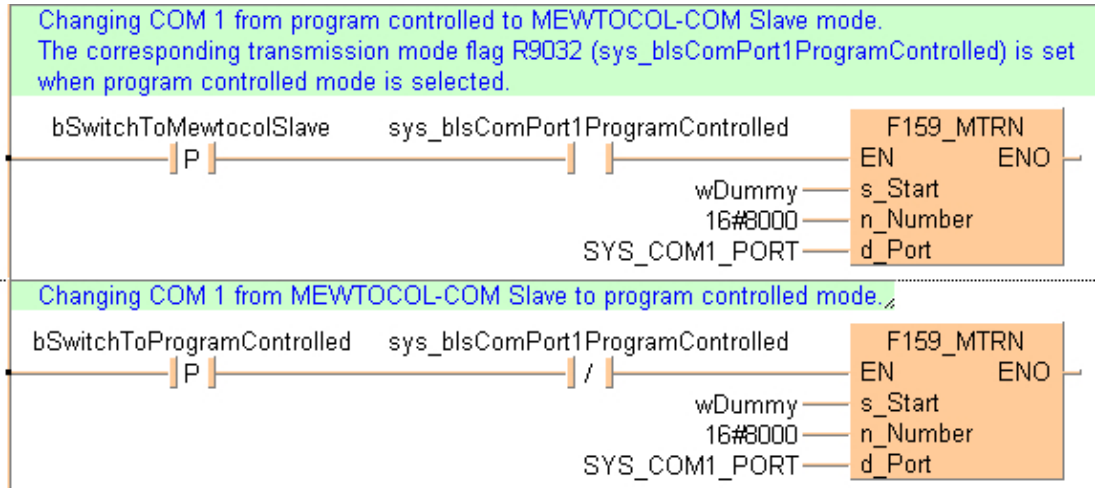
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSwitchToMewtocolSlave | BOOL | FALSE |
| 1 | VAR | wDummy | WORD | 0 |
| 2 | VAR | bSwitchToProgramControlled | BOOL | FALSE |

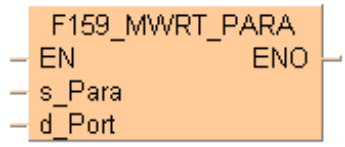
### LD body



# F159_MWRT_PARA

Set communication parameters in RUN mode

Communication parameters in the predefined DUT **MCU_PARA_DUT** are written to the specified port of a Multi-Communication Unit.




## Parameters

Input

**s_Para** (MCU_PARA_DUT)

Communication parameters defined in the predefined DUT

**d_Port** (WORD, INT, UINT)

Specification of slot number (high byte) and port number (low byte) of the MCU to which the data is transmitted.

## Remarks

To enable any settings made with the MCU Configurator, set DIP switches 3 and 4 of the MCU to ON for COM port 1. For COM port 2 set switches 7 and 8 to ON.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if the MCU unit does not exist in the specified slot or zero bytes should be sent.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if the MCU unit does not exist in the specified slot or zero bytes should be sent.

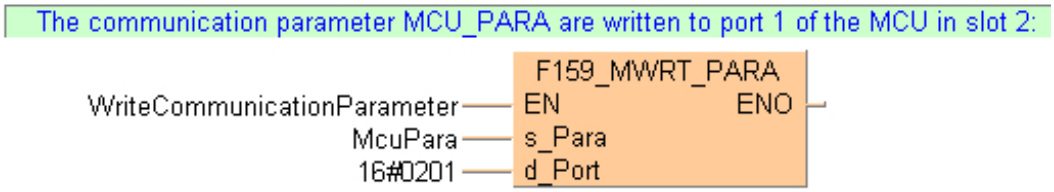
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | WriteCommunicationParameter | BOOL | FALSE |
| 1 | VAR | McuPara | MCU_PARA_DUT | |

## LD body



### 10.1.9 Data exchange with flexible network

# FNS_InitConfigDataTable

Configure data for FP-FNS blocks

The **FNS_InitConfigDataTable** function creates a **ConfigDataTable** from the variable **ProcessDataTable**, which can be a single-element data type or a multi-element data type. This **ConfigDataTable** is necessary to configure the FP-FNS block using the function block **FNS_ProfibusDP**, **FNS_DeviceNet**, **FNS_CANopen** and **FNS_ProfinetIO**.


## Parameters

Input

**ProcessDataTable** (INT, WORD, DINT, DWORD, REAL, TIME) and ARRAYS of these types

Input and output of process data variables

Output

**ConfigDataTable** (ARRAY of WORD)

Configuration data for FP-FNS blocks. The array size of the variable **ConfigDataTable** has to correspond to the number of elements of the **ProcessDataTable** variable.

## Remarks

- Make sure that the size of the variable **ConfigDataTable** corresponds to the structure of the variable **ProcessDataTable**, e.g. if **ProcessDataTable** consists of three entries, then **ConfigDataTable** has to be an "Array[0..2] of WORD" with a size that matches the number of entries. If **ProcessDataTable** has only one entry (e.g. WORD), then **ConfigDataTable** has to be an "Array[0..0] of WORD" (with size 1).
- Allowed data types for the input of the **FNS_InitConfigDataTable** are all 16-bit (INT, WORD), 32-bit (DINT, DWORD, TIME (32 bits), REAL) and 64-bit variables or arrays of them. 64-bit variables are defined as 2-dimensional arrays, e.g. "Array[0..0,0..3] of INT" is a 64-bit variable, while "Array[0..3] of INT" represents an array with four elements of 16-bit variables.
- The data types BOOL, STRING and arrays of these types are NOT allowed at the input of the function **FNS_InitConfigDataTable**.
- The output **ConfigDataTable** of the function must be an array of WORD.

## ProcessDataTable

- The following syntax table shows how to declare 16-bit, 32-bit and 64-bit variables and arrays thereof when using them as **ProcessDataTable** input for the **FNS_InitConfigDataTable** function.

| Input Data type | Size of Input | Comment |
|---|---|---|
| **INT, WORD** | 16-bit | |
| **DINT, WORD, REAL, TIME** | 32-bit | |
| **Array[0..0,0..3] of INT**<br>**Array[0..0,0..3] of WORD** | 64-bit | 2-dimensional array;<br>size of second dimension = 4 |
| **Array[a ..b] of INT/Array[a ..b] of WORD** | Array of 16-bit<br>Size = b-a+1 | 1-dimensional array |
| **Array[a ..b] of DINT/Array[a ..b] of DWORD/**<br>**Array[a ..b] of REAL/Array[a ..b] of TIME** | Array of 32-bit<br>Size = b-a+1 | 1-dimensional array |
| **Array[0..x,0..3] of INT**<br>**Array[0..x,0..3] of WORD** | Array of 64-bit<br>Size = x+1 | 2-dimensional array;<br>size of second dimension = 4 |

- For each element of the DUT applied at **ProcessDataTable**, an element in the WORD array at **ConfigDataTable** is generated according to the following formula:

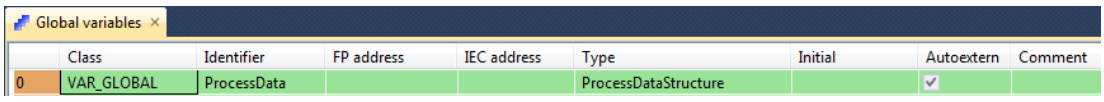
```
enum class enumFnsDataTypeCode : int
      {
            INVALID     = -1,
            SIGNED_16   = 0x02,
            SIGNED_32   = 0x03,
            UNSIGNED_16 = 0x05,
            UNSIGNED_32 = 0x06,
            UNSIGNED_64 = 0x11,
            REAL_32     = 0x12
      };
```

  - Low byte: data type code required for FNS
  - High byte: number of elements in case of an array

## Example

## Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.




DUT

In this example, the variable **ProcessData** is a DUT of the type **ProcessDataStructure** with the following structure:

ProcessDataStructure

| | Identifier | Type | Initial |
|---|---|---|---|
| 1 | Data1_16bits | INT | 0 |
| 2 | Data2_32bits | DWORD | 0 |
| 3 | Data3_arraysize2_64bits | ARRAY [0..1,0..3] OF INT | [8(0)] |

As the DUT has three entries, the output variable **ConfigData** has to be an array of WORD with a size of three (e.g.: Array [0..2] of WORD).

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

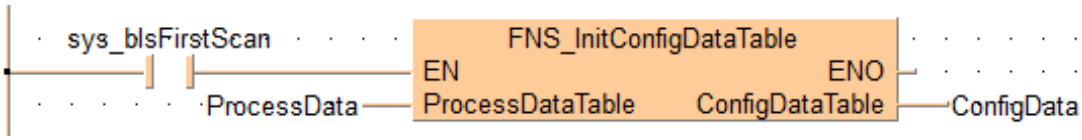
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | ConfigData | ARRAY [0..2] OF WORD | [3(0)] |
| 1 | VAR_EXTERNAL | ProcessData | ProcessDataStructure | |

The size of the variable **ConfigDataTable** has to correspond to the number of entries of the input variable **ProcessData**.

### POU body

When **sys_bIsFirstScan** is TRUE, i.e. in the first cycle, the function is executed. The value of the variable **ConfigData** corresponds to the structure of the input variable **ProcessData**, its number and type of elements.

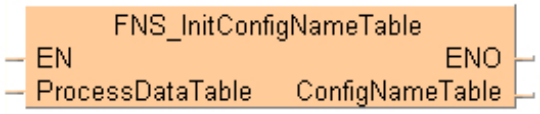
### LD body



# FNS_InitConfigNameTable

Configure data for FP-FNS blocks

The **FNS_InitConfigNameTable** function creates a **ConfigNameTable** from the variable **ProcessDataTable**, which can be a single-element data type or a multi-element data type.




## Parameters

Input

**ProcessDataTable** (INT, WORD, DINT, DWORD, REAL, TIME) and ARRAYS of these types

Input and output of process data variables

Output

**ConfigNameTable** ARRAY of WORD

The output is an array of STRING containing the member names of the DUT applied at **ProcessDataTable**.

- The array size of the variable **ConfigNameTable** has to correspond to the number of elements of the **ProcessDataTable** variable.
- The string length of the array elements must be greater than or equal to the maximum length of the member names.

## Remarks

- Make sure that the size of the variable **ConfigNameTable** corresponds to the structure of the variable **ProcessDataTable**, e.g. if **ProcessDataTable** consists of three entries, then **ConfigNameTable** has to be an "Array[0..2] of WORD" with a size that matches the number of entries. If **ProcessDataTable** has only one entry (e.g. WORD), then **ConfigNameTable** has to be an "Array[0..0] of WORD" (with size 1).
- Allowed input data types are all 16-bit (INT, WORD), 32-bit (DINT, DWORD, TIME (32 bits), REAL) and 64-bit variables or arrays of them. 64-bit variables are defined as 2-dimensional arrays, e.g. "Array[0..0,0..3] of INT" is a 64-bit variable, while "Array[0..3] of INT" represents an array with four elements of 16-bit variables.
- The data types BOOL, STRING and arrays of these types are NOT allowed at the input of the function **FNS_InitConfigDataTable**.

### ProcessDataTable

The following syntax table shows how to declare 16-bit, 32-bit and 64-bit variables and arrays thereof when using them as **ProcessDataTable** input.

| Input Data type | Size of Input | Comment |
|---|---|---|
| **INT, WORD** | 16-bit | |
| **DINT, WORD, REAL, TIME** | 32-bit | |
| **Array[0..0,0..3] of INT**<br>**Array[0..0,0..3] of WORD** | 64-bit | 2-dimensional array;<br>size of second dimension = 4 |
| **Array[a ..b] of INT/Array[a ..b] of WORD** | Array of 16-bit<br>Size = b-a+1 | 1-dimensional array |
| **Array[a ..b] of DINT/Array[a ..b] of DWORD/**<br>**Array[a ..b] of REAL/Array[a ..b] of TIME** | Array of 32-bit<br>Size = b-a+1 | 1-dimensional array |
| **Array[0..x,0..3] of INT**<br>**Array[0..x,0..3] of WORD** | Array of 64-bit<br>Size = x+1 | 2-dimensional array;<br>size of second dimension = 4 |

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | adiDInt1 | ARRAY [2..3] OF DINT | [2(0)] |
| 2 | VAR | arReal1 | ARRAY [0..1,1..9] OF REAL | [18(0.0)] |
| 3 | VAR | aui1 | ARRAY [1..15] OF UINT | [15(0)] |
| 4 | VAR | dwWord1 | DWORD | 0 |
| 5 | VAR | iInt1 | INT | 0 |
| 6 | VAR | udiUDInt1 | UDINT | 0 |

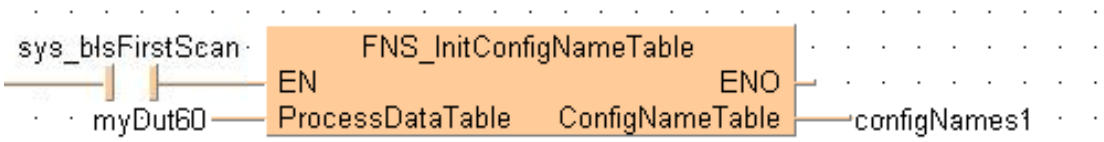
The size of the variable **configNames1** has to correspond to the number of entries of the input variable **myDUT60**.

As the DUT has three entries, the output variable **configNames1** has to be an array of WORD with a size of three (e.g.: Array [0..2] of WORD).

### POU body

When **sys_bIsFirstScan** is TRUE, i.e. in the first cycle, the function is executed. The value of the variable **configNames1** corresponds to the structure of the input variable **myDUT60**, its number and type of elements.

### LD body




## 10.2 Getting communication parameters

Use F instructions or system variables to check the PLC's or MCU's communication settings in RUN mode.

Select the appropriate method to get communication parameters in RUN mode:

| Parameters | | Method | CPU | MCU |
|---|---|---|---|---|
| Communication mode | Program controlled<br>MEWTOCOL-COM Master/Slave | IsProgramControlled | ● | ● |
| | | **sys_bIsToolPortProgramControlled** | ● | |
| | | **sys_bIsComPort1ProgramControlled** | ● | |
| | | **sys_bIsComPort2ProgramControlled** | ● | |
| | | F161_MRD_STATUS | | ● |
| | “PLC link” | IsPlcLink | ● | ● |
| | | **sys_bIsComPort1PlcLink** | ● | |
| | | F161_MRD_STATUS | | ● |
| Other parameters<br>(e.g. baud rate, station number, start and end code) | | F161_MRD_PARA | | ● |
| | | F161_MRD_STATUS | | ● |

# IsMasterCommunication

Evaluate "IsMasterCommunication" flag

This instruction returns the value of the "IsMasterCommunication" flag.


## Parameters

Input

**Port** (WORD, INT, UINT)

Communication port: 1, 2
Ethernet user connection: port 1–216

## Remarks

Evaluation of the "IsMasterCommunication" flag. The flag can be evaluated using one of the following system variables:

- **sys_bIsComPort0MasterCommunication**
- **sys_bIsComPort1MasterCommunication**
- **sys_bIsComPort2MasterCommunication**
- **sys_bIsEthernetUserConnection1 ... 216MasterCommunication**

  For communication ports not supporting the master function, the flag is always TRUE.

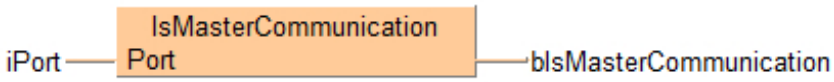
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iPort | INT | 0 |
| 1 | VAR | bIsMasterCommunication | BOOL | FALSE |

### LD body


IsMasterCommunication
iPort
Port
blsMasterCommunication

# IsPlcLink

Evaluate "PLC Link" flag

This instruction returns the value of the "PLC Link" flag. The flag is TRUE if the communication port of the PLC has been set to PLC Link mode.


## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iPort | INT | 0 |
| 1 | VAR | PlcLink | BOOL | FALSE |

### LD body

# IsProgramControlled

Evaluate communication mode flag

This instruction returns the value of the communication mode flag. The communication mode flag is TRUE if the communication port of the PLC has been set to program controlled mode. It is FALSE if it has been set to Modbus/MEWTOCOL mode.


## Parameters

Input

**Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iPort | INT | 0 |
| 1 | VAR | bProgramControlled | BOOL | FALSE |

### LD body

### 10.2.4 FP instructions

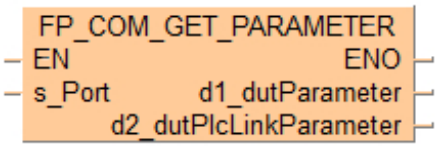
Tip

Advantages of FP instructions

# FP_COM_GET_PARAMETER

Get communication parameters from CPU/SCU/MCU port

This FP instruction reads the parameters of the communication port specified by **s_Port** of the CPU unit and stores the result in the DUT **d1_dutParameter**.




## Parameters

Input

**s_Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

Output

**d1_dutParameter** (**FP_COM_PARAMETER_DUT**)

CPU/SCU/MCU parameters to be read

**d2_dutPLCLinkParameter** (**FP_COM_PLCLINK_PARAMETER_DUT**)

Starting address of parameters to be stored

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

### POU header

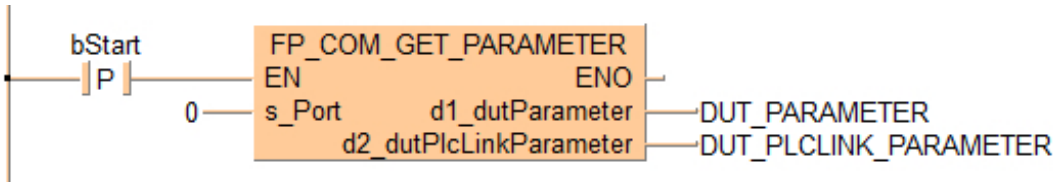
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | DUT_PLCLINK_PARAMETER | FP_COM_PLCLINK_PARAMETER_DUT | |
| 2 | VAR | DUT_PARAMETER | FP_COM_PARAMETER_DUT | |

### POU body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out.

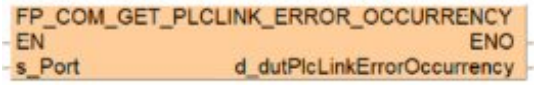
### LD body



# FP_COM_GET_PLCLINK_ERROR_OCCURRENCY

Get PLC link error frequency

This FP instruction reads the PLC link error frequency information for the communication port specified by **s_Port** of the CPU unit and stores the result in the DUT **d_dutPlcLinkErrorOccurrency**.




## Parameters

Input

**s_Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

Output

**d_dutPlcLinkErrorOccurrency** (**FP_COM_PLCLINK_ERROR_OCCURRENCY_DUT**)

Stores the number of occurrences in the **FP_COM_PLCLINK_ERROR_OCCURRENCY_DUT**

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

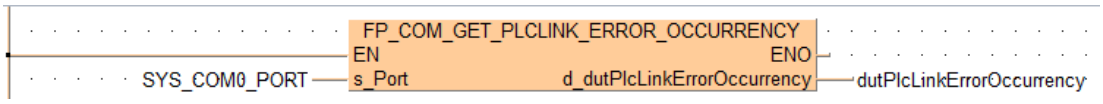
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type |
|---|---|---|---|
| 1 | VAR | dutPlcLinkErrorOccurrency | FP_COM_PLCLINK_ERROR_OCCURRENCY_DUT |

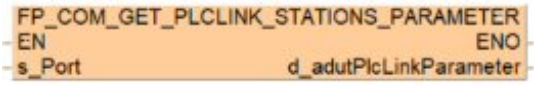
## LD body



# FP_COM_GET_PLCLINK_STATIONS_PARAMETER

Get PLC link settings

This FP instruction monitors the PLC link settings of all PLC link stations connected to the communication port specified by **s_Port** of the CPU unit and stores the result in the DUT **d_adutPlcLinkParameter**.




## Parameters

Input

**s_Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

Output

**d_adutPlcLinkParameter** (**FP_COM_PLCLINK_PARAMETER_DUT**)

Array of data unit types for the PLC link settings of the PLC link stations connected to the specified COM port (ARRAY [1..16] OF FP_COM_PLCLINK_PARAMETER_DUT)

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

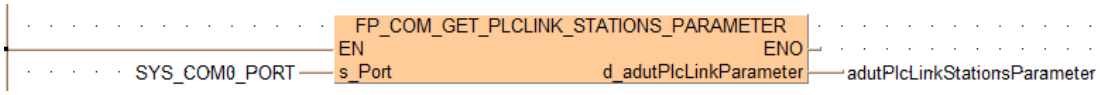
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type |
|---|---|---|---|
| 1 | VAR | adutPlcLinkStationsParameter | ARRAY [1..16] OF FP_COM_PLCLINK_PARAMETER_DUT |

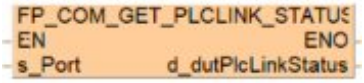
## LD body



# FP_COM_GET_PLCLINK_STATUS

Get PLC link status

This FP instruction reads the PLC link status flag information for the communication port specified by **s_Port** of the CPU unit and stores the result in the DUT **d_dutPlcLinkStatus**.




## Parameters

Input

**s_Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

Output

**d_dutPlcLinkStatus** (**FP_COM_PLCLINK_STATUS_DUT**)

Stores the PLC link status information in the **FP_COM_PLCLINK_STATUS_DUT**

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

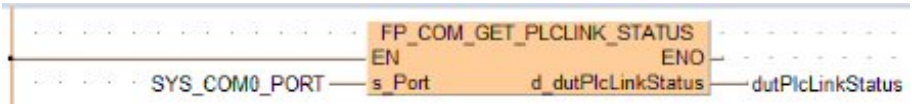
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type |
|---|---|---|---|
| 1 | VAR | dutPlcLinkStatus | FP_COM_PLCLINK_STATUS_DUT |

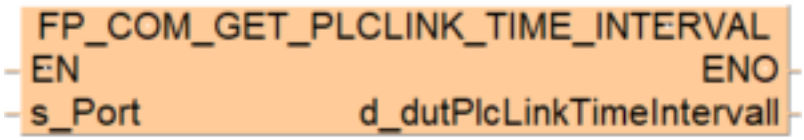
### LD body


FP_COM_GET_PLCLINK_STATUS
EN
ENO
SYS_COM0_PORT
s_Port
d_dutPlcLinkStatus
dutPlcLinkStatus

# FP_COM_GET_PLCLINK_TIME_INTERVAL

Get PLC link time interval

This FP instruction reads the PLC link time interval information for the communication port specified by **s_Port** of the CPU unit and stores the result in the DUT **d_dutPlcLinkTimeInterval**.




## Parameters

Input

**s_Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

Output

**d_dutPlcLinkTimeInterval** (**FP_COM_PLCLINK_TIME_INTERVAL_DUT**)

Stores the PLC link time interval values in the **FP_COM_PLCLINK_TIME_INTERVAL_DUT**

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

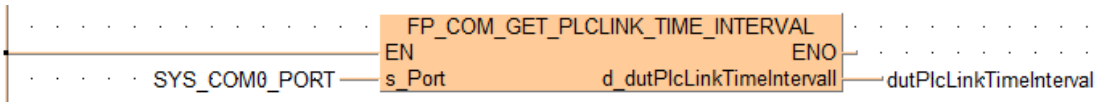
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type |
|---|---|---|---|
| 1 | VAR | dutPlcLinkTimeInterval | FP_COM_PLCLINK_TIME_INTERVAL_DUT |

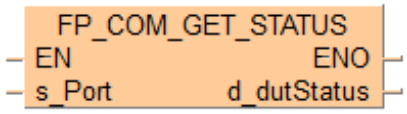
## LD body



# FP_COM_GET_STATUS

Read communication port status

This FP instruction gets the status of the communication port specified by **s_Port** of the CPU unit and stores the result in the DUT **d_dutStatus**.




## Parameters

Input

**s_Port** (WORD, INT, UINT)

Specifies the communication ports depending on the PLC type:

- COM port e.g. **SYS_COM0_PORT**
- Ethernet port e.g. **SYS_ETHERNET_USER_CONNECTION_1**
- MCU/SCU e.g. 16#xx01 (xx = slot number) in COM01

Output

**d_dutStatus** (**FP_COM_STATUS_DUT**)

Stores the CPU/SCU/MCU status information in the **FP_COM_STATUS_DUT**

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## POU header

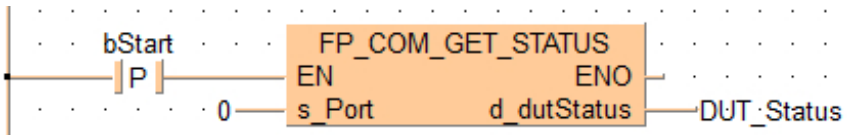
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | DUT_SCUStatus | FP_SCU_STATUS_DUT | |

## POU body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out.

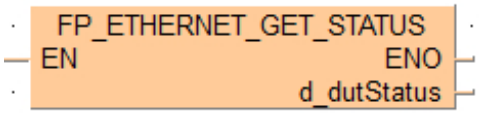
## LD body



# FP_ETHERNET_GET_STATUS

Read ET-LAN status

This FP instruction gets the status information of all Ethernet connections.




## Parameters

Output

**d_dutStatus** (**FP_ETHERNET_STATUS_DUT**)

Stores the status information of all Ethernet connections in the **FP_ETHERNET_STATUS_DUT**:

- Connection status of all connections
- OPEN status
- OPEN error status
- No. of connections in-progress in the FTP server

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the destination range is out of the accessible range.
- if the area storing parameters which is specified by **d_dutStatus** is invalid.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the destination range is out of the accessible range.
- if the area storing parameters which is specified by **d_dutStatus** is invalid.

## Example

## POU header

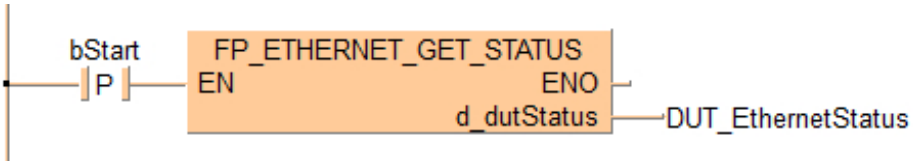
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | DUT_EthernetStatus | FP_ETHERNET_STATUS_DUT | |
| 2 | VAR | uiResult | UINT | 0 |

### POU body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out.

### LD body



# FP_ETHERNET_PING

Request to send PING

This FP instruction sends a PING to the destination IP address of the connection specified by **sPort**. The number of times to send PING is specified by **n_Requests**. Please ensure the suitable parameters are also set in the project navigator under “System registers” > “Ethernet” > “User connections”.


## Parameters

Input

**sPort** (WORD, INT, UINT)

Ethernet port on CPU (FP7 E types):
**SYS_ETHERNET_USER_CONNECTION_1**–**SYS_ETHERNET_USER_CONNECTION_216**

**n_Requests** (WORD, INT, UINT)

Number of times to send PING
Values: 1–10

Output

**d** (**FP_ETHERNET_PING_DUT**)

Stores the result of PING in the DUT **FP_ETHERNET_PING_DUT**

## Remarks

- This instruction is dedicated to ET-LAN.
- The timeout period for one PING response is one second (fixed).
- The size of sent/received data is 56 bytes (fixed).
- When a destination IP address is not specified, an error occurs.
- Use the instructions **FP_IPV4_GET_CONNECTION** or **FP_IPV6_GET_CONNECTION** to check the destination IP address for the PING.
- When Ethernet is being initialized while sending PING, 0 is put in all elements of **FP_ETHERNET_PING_DUT**.

## Example

### POU header

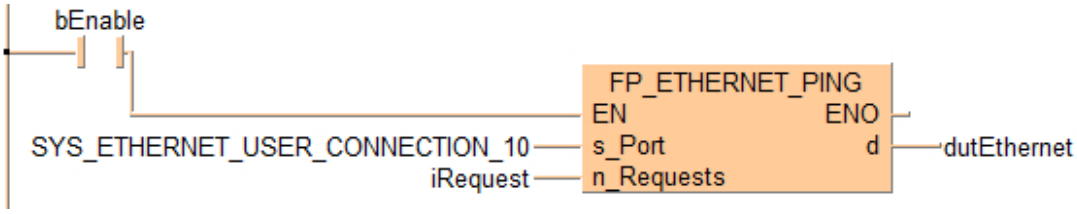
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable | BOOL | FALSE |
| 1 | VAR | iRequest | INT | 0 |
| 2 | VAR | dutEthernet | FP_ETHERNET_PING_DUT | |

### POU body

When the variable **bEnable** is set to TRUE, the function is executed.

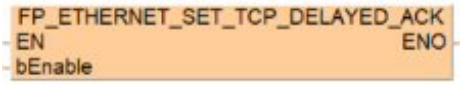
### LD body



# FP_ETHERNET_SET_TCP_DELAYED_ACK

Enable or disable the TCP delayed acknowledgement

This instruction enables/disables TCP delayed acknowledgement for the FP7. Disabling TCP delayed acknowledgement for the FP7 speeds up communication if the other device is set to use TCP delayed acknowledgement.




## Parameters

Input

**bEnable** (BOOL)

TRUE: Ethernet TCP delayed acknowledgement enabled
FALSE: Ethernet TCP delayed acknowledgement disabled

## Remarks

- This instruction cannot be executed while the built-in Ethernet is being initialized. Before you execute the instruction, make sure that **sys_bIsEthernetIPAddressAssigned** is set to FALSE.
- Set this instruction to be executed only one time after switching to “RUN mode”.
- Upper and lower case characters can be used for operands for which a character constant can be specified. "Abcd", "ABCD" and "abcd" are synonymous, however, file names are case-sensitive.
- When the power is turned ON, TCP delayed acknowledgement is enabled. After this instruction is executed by setting **bEnable** to FALSE, TCP delayed acknowledgement is disabled and the FP7 sends acknowledgements without delay.
- Use the system variable **sys_bIsEthernetTCPDelayedAckEnabled** to check whether TCP delayed acknowledgement is enabled (TRUE) or disabled (FALSE).

## Error flags

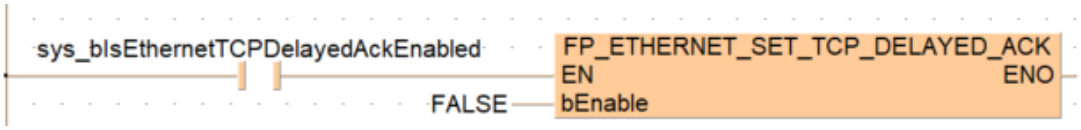
**sys_bIsCarry** (turns to TRUE for one scan)

if the instruction is executed during the initialization of Ethernet, **sys_iEthernetConnectionErrorCode** is set to "11: Ethernet is being initialized".

## Example

### LD body

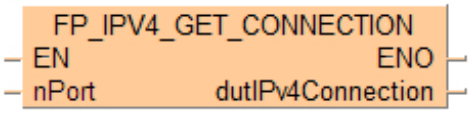
When the variable **sys_bIsEthernetTCPDelayedAckEnabled** is set to TRUE, the function is carried out.



# FP_IPV4_GET_CONNECTION

Get connection parameters

This FP instruction reads the parameters from the Ethernet connection and writes the values into the DUT **FP_IPv4_CONNECTION_DUT**.




## Parameters

Input

**nPort** (WORD, INT, UINT)

Ethernet port on CPU (FP7 E types):
**SYS_ETHERNET_USER_CONNECTION_1**–**SYS_ETHERNET_USER_CONNECTION_216**

Output

**dutIPv4Connection** (**FP_IPv4_CONNECTION_DUT**)

Parameters of the Ethernet connection

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

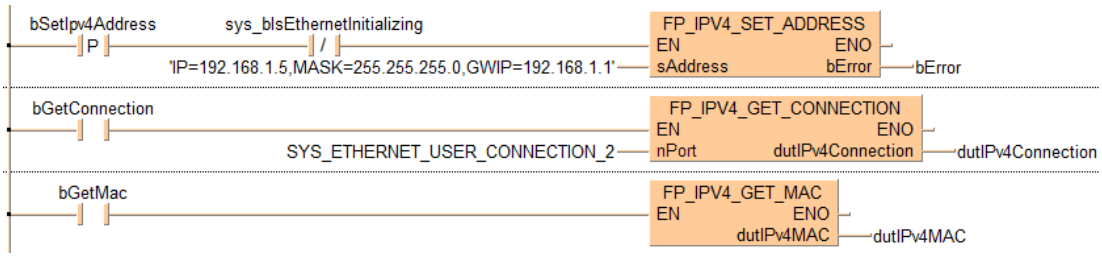
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetIpv4Address | BOOL | FALSE |
| 1 | VAR | bGetConnection | BOOL | FALSE |
| 2 | VAR | bGetMac | BOOL | FALSE |
| 3 | VAR | dutIPv4Connection | FP_IPv4_CONNECTION_DUT | |
| 4 | VAR | dutIPv4MAC | FP_IPv4_MAC_DUT | |

### POU body

This example sets the Ethernet connection parameters specified by **s_Address**. Please ensure the suitable parameters are also set in the project navigator under “System registers” > “Ethernet” > “User connections”. When **bGetConnection** is set to TRUE, the connection parameters are entered into the DUT **FP_IPv4_CONNECTION_DUT**.When

**bGetMac** is set to TRUE, the parameters of the MAC address are entered into the DUT **FP_IPv4_MAC_DUT**.

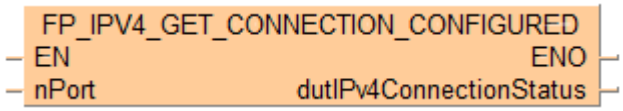
## LD body



# FP_IPV4_GET_CONNECTION_CONFIGURED

Return the IPv4 connection parameters set by system registers

This FP instruction returns the connection parameters of an IPv4 connection as it has been configured in system registers or by **FP_ETHERNET_CONNECTION_SET**.




Input

**nPort** (WORD, INT, UINT)

Ethernet port on CPU (FP7 E types):
**SYS_ETHERNET_USER_CONNECTION_1**–**SYS_ETHERNET_USER_CONNECTION_216**

Output

**dutIPv4ConnectionStatus** (**FP_IPv4_CONNECTION_STATUS_DUT**)

Parameters of the Ethernet connection

| Elements of the DUT (identifiers): | Data type | With/without connection established |
|---|---|---|
| **IPv4** | Array[0..3] OF WORD | IP address in Internet protocol version 4 format |
| **SubnetMask** | Array[0..3] OF WORD | Subnet mask |
| **DefaultGateway** | Array[0..3] OF WORD | Default gateway |
| **wHomePortNumber** | WORD | Number of the home port |
| **wDestIPAddressType** | Array[0..3] OF WORD | Address type of destination IP address |
| **IPv4Dest** | Array[0..3] OF WORD | Destination IP address in Internet protocol version 4 format |
| **wDestPortNumber** | WORD | Number of the destination port |

## Example

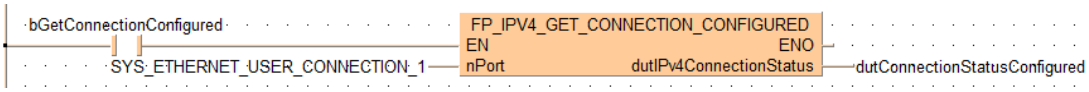
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | dutConnectionStatusConfigured | FP_IPv4_CONNECTION_STATUS_DUT | |
| 1 | VAR | bGetConnectionConfigured | BOOL | FALSE |

## LD body

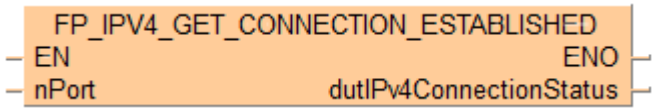
When the variable **bGetConnectionConfigured** is set to TRUE, the function is carried out.



# FP_IPV4_GET_CONNECTION_ESTABLISHED

Return the IPv4 connection parameters

This FP instruction returns the connection parameters of an IPv4 connection. Which connection parameters are returned depends on whether the connection has been established or not.




Input

**nPort** (WORD, INT, UINT)

Ethernet port on CPU (FP7 E types):
**SYS_ETHERNET_USER_CONNECTION_1**–**SYS_ETHERNET_USER_CONNECTION_216**

Output

**dutIPv4ConnectionStatus** (**FP_IPv4_CONNECTION_STATUS_DUT**)

Parameters of the Ethernet connection

| Elements of the DUT (identifiers): | Data type | With connection established | Without connection established |
|---|---|---|---|
| **IPv4** | Array[0..3] OF WORD | IP address in Internet protocol version 4 format | IP address in Internet protocol version 4 format |
| **SubnetMask** | Array[0..3] OF WORD | Subnet mask | Subnet mask |
| **DefaultGateway** | Array[0..3] OF WORD | Default gateway | Default gateway |
| **wHomePortNumber** | WORD | Number of the home port | 0 |
| **wDestIPAddressType** | Array[0..3] OF WORD | Address type of destination IP address | Address type of destination IP address |
| **IPv4Dest** | Array[0..3] OF WORD | Address type of destination IP address | Address type of destination IP address |
| **wDestPortNumber** | WORD | Destination IP address in Internet protocol version 4 format | 0 |

## Example

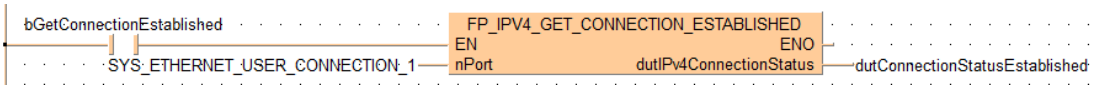
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | dutConnectionStatusEstablished | FP_IPv4_CONNECTION_STATUS_DUT | |
| 1 | VAR | bGetConnectionEstablished | BOOL | FALSE |

### LD body

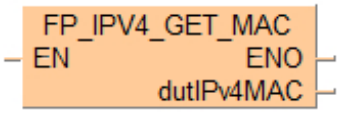
When the variable **bGetConnectionEstablished** is set to TRUE, the function is carried out.



# FP_IPV4_GET_MAC

Get parameters of MAC address

This FP instruction reads the parameters of the MAC address used by the Ethernet connection and writes the values into the DUT **FP_IPv4_MAC_DUT**.




## Parameters

Output

**dutIPv4MAC** (**FP_IPv4_MAC_DUT**)

Parameters of MAC address

## Example

## POU header

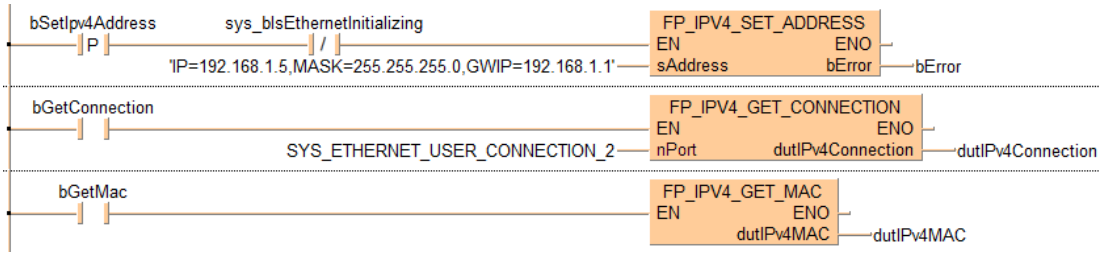
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetIpv4Address | BOOL | FALSE |
| 1 | VAR | bGetConnection | BOOL | FALSE |
| 2 | VAR | bGetMac | BOOL | FALSE |
| 3 | VAR | dutIPv4Connection | FP_IPv4_CONNECTION_DUT | |
| 4 | VAR | dutIPv4MAC | FP_IPv4_MAC_DUT | |

## POU body

This example sets the Ethernet connection parameters specified by **s_Address**. Please ensure the suitable parameters are also set in the project navigator under “System registers” > “Ethernet” > “User connections”. When **bGetConnection** is set to TRUE, the connection parameters are entered into the DUT **FP_IPv4_CONNECTION_DUT**. When **bGetMac** is set to TRUE, the parameters of the MAC address are entered into the DUT **FP_IPv4_MAC_DUT**.

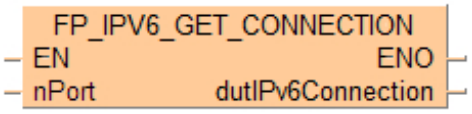
## LD body


bSetIpv4Address
P
sys_blsEthernetInitializing
/
'IP=192.168.1.5,MASK=255.255.255.0,GWIP=192.168.1.1'
FP_IPV4_SET_ADDRESS
EN
ENO
sAddress
bError
bError
bGetConnection
FP_IPV4_GET_CONNECTION
EN
ENO
SYS_ETHERNET_USER_CONNECTION_2
nPort
dutIPv4Connection
dutIPv4Connection
bGetMac
FP_IPV4_GET_MAC
EN
ENO
dutIPv4MAC
dutIPv4MAC

# FP_IPV6_GET_CONNECTION

Get connection parameters

This FP instruction reads the parameters from the Ethernet connection and writes the values into the DUT **FP_IPV6_CONNECTION_DUT**.




## Parameters

Input

**nPort** (WORD, INT, UINT)

Ethernet port on CPU (FP7 E types):
**SYS_ETHERNET_USER_CONNECTION_1**–**SYS_ETHERNET_USER_CONNECTION_216**

Output

**dutIPv6Connection** (**FP_IPV6_CONNECTION_DUT**)

Parameters of the Ethernet connection

## Example

## POU header

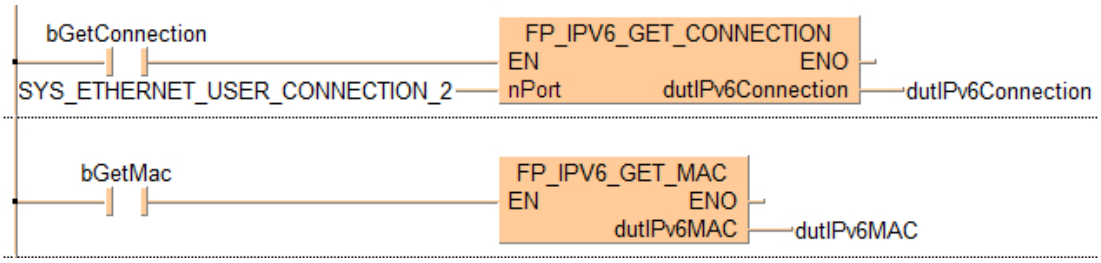
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bGetConnection | BOOL | FALSE |
| 1 | VAR | dutIPv6Connection | FP_IPv6_CONNECTION_DUT | |
| 2 | VAR | bGetMac | BOOL | FALSE |
| 3 | VAR | dutIPv6MAC | FP_IPv6_MAC_DUT | |

## POU body

When the variables **bGetConnection** and **bGetMac** are set to TRUE, the functions are carried out.

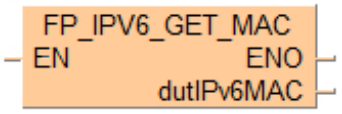
## LD body


bGetConnection
FP_IPV6_GET_CONNECTION
EN
ENO
SYS_ETHERNET_USER_CONNECTION_2
nPort
dutIPv6Connection
dutIPv6Connection
bGetMac
FP_IPV6_GET_MAC
EN
ENO
dutIPv6MAC
dutIPv6MAC

# FP_IPV6_GET_MAC

Get parameters of MAC address

This FP instruction reads the parameters of the MAC address used by the Ethernet connection and writes the values into the DUT **FP_IPV6_MAC_DUT**.




## Parameters

Output

**dutIPv6MAC** (**FP_IPV6_MAC_DUT**)

Parameters of MAC address

## Example

## POU header

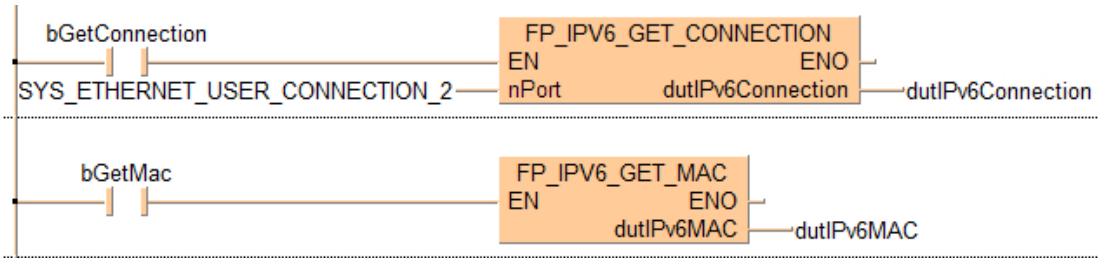
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bGetConnection | BOOL | FALSE |
| 1 | VAR | dutIPv6Connection | FP_IPv6_CONNECTION_DUT | |
| 2 | VAR | bGetMac | BOOL | FALSE |
| 3 | VAR | dutIPv6MAC | FP_IPv6_MAC_DUT | |

## POU body

When the variables **bGetConnection** and **bGetMac** are set to TRUE, the functions are carried out.

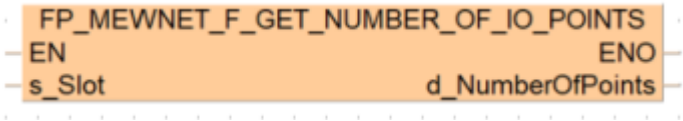
## LD body


bGetConnection
FP_IPV6_GET_CONNECTION
EN
ENO
SYS_ETHERNET_USER_CONNECTION_2
nPort
dutIPv6Connection
dutIPv6Connection
bGetMac
FP_IPV6_GET_MAC
EN
ENO
dutIPv6MAC
dutIPv6MAC

# FP_MEWNET_F_GET_NUMBER_OF_IO_POINTS

Read MEWNET-F number of I/O points

This FP instruction reads the number of I/O points of the MEWNET-F unit in the slot number specified by **s_Slot**.




Input

**s_Slot** (ANY16)

Slot number of expansion unit

Output

**d_NumberOfPoints** (ANY16)

Number of I/O points

## Example

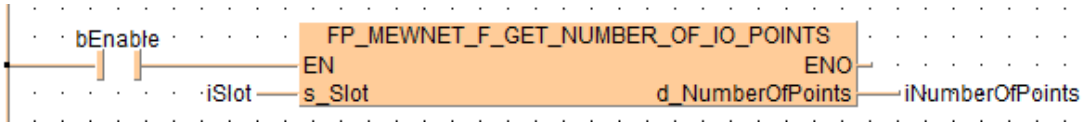
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | iSlot | INT | 0 |
| 3 | VAR | iNumberOfPoints | INT | 0 |

### LD body

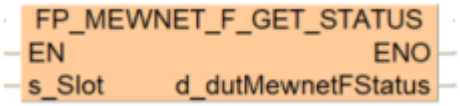
When the variable **bEnable** is set to TRUE, the function is executed.



# FP_MEWNET_F_GET_STATUS

Read MEWNET-F status information

This FP instruction reads the MEWNET-F status information of the MEWNET-F unit in the slot number specified by **s_Slot**.




Input

**s_Slot** (ANY16)

Slot number of expansion unit

Output

**d_dutMewnetFStatus**

Stores the status information in the **FP_MEWNET_F_STATUS_DUT**.

## Example

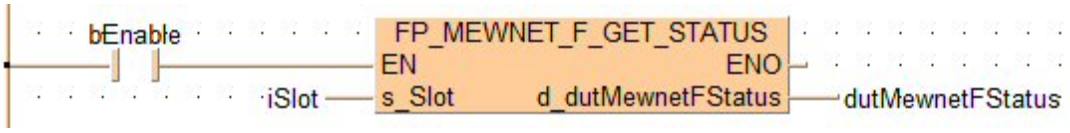
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | iSlot | INT | 0 |
| 3 | VAR | dutMewnetFStatus | FP_MEWNET_F_STATUS_DUT | |

### LD body

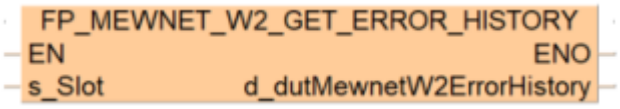
When the variable **bEnable** is set to TRUE, the function is executed.



# FP_MEWNET_W2_GET_ERROR_HISTORY

Read MEWNET-W2 system register area for errors

This FP instruction reads the MEWNET-W2 system register area for storing errors that have occurred on the MEWNET-W2 unit in the slot number specified by **s_Slot**.




Input

**s_Slot** (ANY16)

Slot number of expansion unit

Output

**d_dutMewnetW2ErrorHistory**

Stores the error history in the **FP_MEWNET_W2_ERROR_HISTORY_DUT**.

## Example

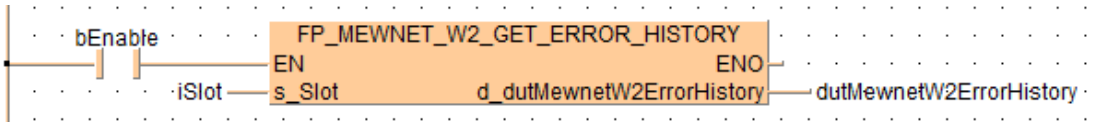
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | iSlot | INT | 0 |
| 3 | VAR | dutMewnetW2ErrorHistory | FP_MEWNET_W2_ERROR_HISTORY_DUT | |

### LD body

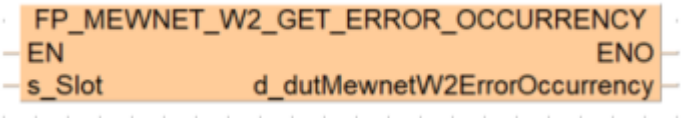
When the variable **bEnable** is set to TRUE, the function is executed.



# FP_MEWNET_W2_GET_ERROR_OCCURRENCY

Get number of error occurrences on MEWNET-W2 units by error type

This FP instruction reads the number of times an error has occurred on the MEWNET-W2 unit in the slot number specified by **s_Slot**.




Input

**s_Slot** (ANY16)

Slot number of expansion unit

Output

**d_dutMewnetW2ErrorOccurrency**

Stores the number of error occurrences by error type in the **FP_MEWNET_W2_ERROR_OCCURRENCY_DUT**.

## Example

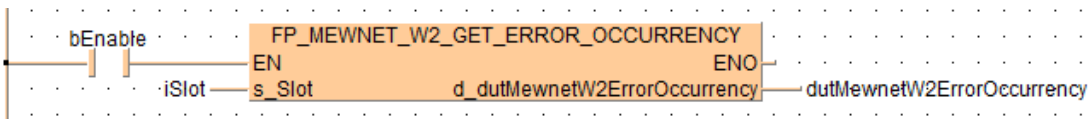
## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | iSlot | INT | 0 |
| 3 | VAR | dutMewnetW2ErrorOccurrency | FP_MEWNET_W2_ERROR_OCCURRENCY_DUT | |

## LD body

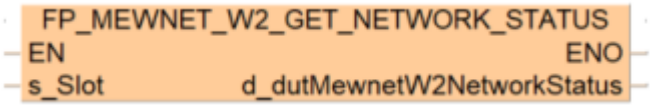
When the variable **bEnable** is set to TRUE, the function is executed.



# FP_MEWNET_W2_GET_NETWORK_STATUS

Get MEWNET-W2 network status information

This instruction gets the MEWNET-W2 network status information about the MEWNET-W2 unit in the slot number specified by **s_Slot**.




Input

**s_Slot** (ANY16)

Slot number of expansion unit

Output

**d_dutMewnetW2NetworkStatus**

Stores the network status information in the **FP_MEWNET_W2_NETWORK_STATUS_DUT**.

## Example

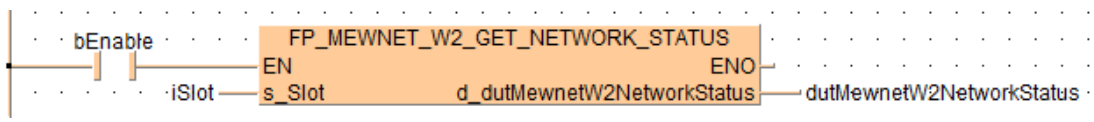
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | iSlot | INT | 0 |
| 3 | VAR | dutMewnetW2NetworkStatus | FP_MEWNET_W2_NETWORK_STATUS_DUT | |

### LD body

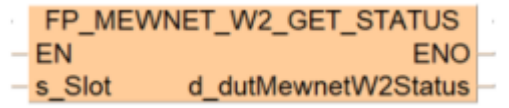
When the variable **bEnable** is set to TRUE, the function is executed.



# FP_MEWNET_W2_GET_STATUS

Read MEWNET-W2 status information

This FP instruction reads the MEWNET-W2 status information of the MEWNET-W2 unit in the slot number specified by **s_Slot**.




Input

**s_Slot** (ANY16)

Slot number of expansion unit

Output

**d_dutMewnetW2Status**

Stores the status information in the **FP_MEWNET_W2_STATUS_DUT**.

## Example

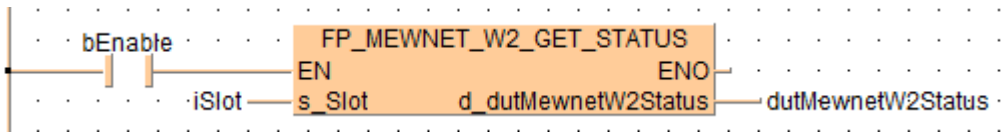
## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | iSlot | INT | 0 |
| 3 | VAR | dutMewnetW2Status | FP_MEWNET_W2_STATUS_DUT | |

## LD body

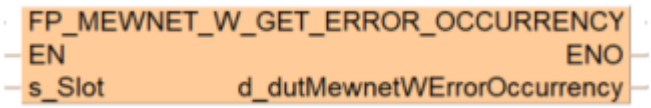
When the variable **bEnable** is set to TRUE, the function is executed.



# FP_MEWNET_W_GET_ERROR_OCCURRENCY

Get number of error occurrences on MEWNET-W units by error type

This FP instruction reads the number of times an error has occurred on the MEWNET-W unit in the slot number specified by **s_Slot**.




Input

**s_Slot** (ANY16)

Slot number of expansion unit

Output

**d_dutMewnetWErrorOccurrency**

Stores the number of error occurrences by error type in the **FP_MEWNET_W_ERROR_OCCURRENCY_DUT**.

## Example

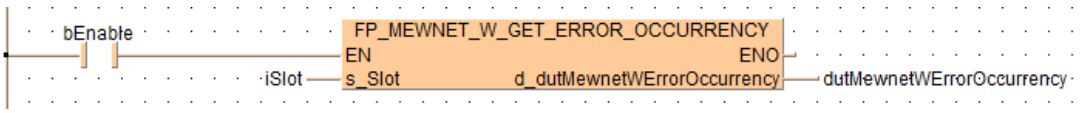
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | iSlot | INT | 0 |
| 3 | VAR | dutMewnetWErrorOccurrency | FP_MEWNET_W_ERROR_OCCURRENCY_DUT | |

### LD body

When the variable **bEnable** is set to TRUE, the function is executed.



# FP_MEWNET_W_GET_NETWORK_STATUS

Get MEWNET-W network status information

This instruction gets the MEWNET-W network status information about the MEWNET-W unit in the slot number specified by **s_Slot**.


Input

**s_Slot** (ANY16)

Slot number of expansion unit

Output

**d_dutMewnetWNetworkStatus**

Stores the network status information in the **FP_MEWNET_W_NETWORK_STATUS_DUT**.

## Example

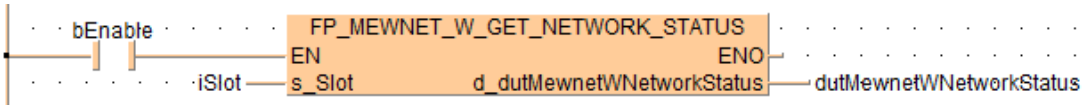
## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | iSlot | INT | 0 |
| 3 | VAR | dutMewnetWNetworkStatus | FP_MEWNET_W_NETWORK_STATUS_DUT | |

## LD body

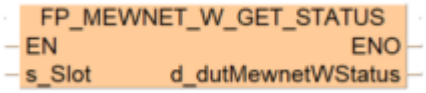
When the variable **bEnable** is set to TRUE, the function is executed.



# FP_MEWNET_W_GET_STATUS

Read MEWNET-W status information

This FP instruction reads the MEWNET-W status information of the MEWNET-W unit in the slot number specified by **s_Slot**.




Input

**s_Slot** (ANY16)

Slot number of expansion unit

Output

**d_dutMewnetWStatus**

Stores the status information in the **FP_MEWNET_W_STATUS_DUT**.

## Example

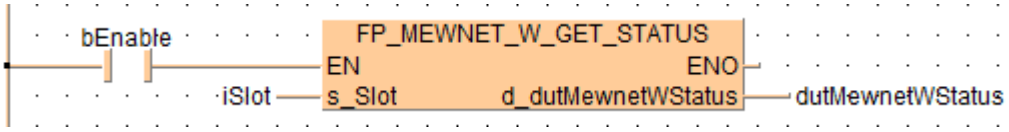
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | iSlot | INT | 0 |
| 3 | VAR | dutMewnetWStatus | FP_MEWNET_W_STATUS_DUT | |

### LD body

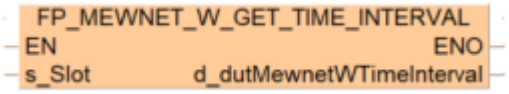
When the variable **bEnable** is set to TRUE, the function is executed.



# FP_MEWNET_W_GET_TIME_INTERVAL

Refresh MEWNET-W monitoring information

This instruction refreshes the MEWNET-W send/receive time interval information for the MEWNET-W unit in the slot specified by **s_Slot**.




Input

**s_Slot** (ANY16)

Slot number of expansion unit

Output

**d_dutMewnetWTimeInterval**

Stores the time interval values in the **FP_MEWNET_W_TIME_INTERVAL_DUT**.

## Example

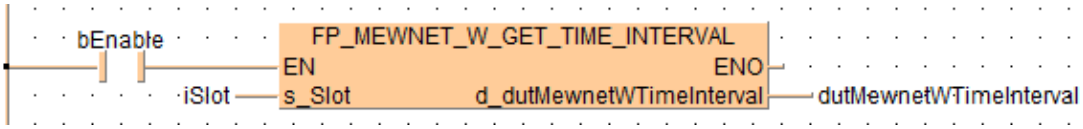
## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | iSlot | INT | 0 |
| 3 | VAR | dutMewnetWTimeInterval | FP_MEWNET_W_TIME_INTERVAL_DUT | |

## LD body

When the variable **bEnable** is set to TRUE, the function is executed.



### 10.2.5 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

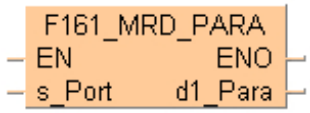
Advantages of IEC instructions

Advantages of FP instructions

# F161_MRD_PARA

Get communication parameters in RUN mode

Communication parameters in the predefined DUT **MCU_PARA_DUT** are received from a port of a Multi-Communication Unit in a certain slot.




## Parameters

Input

**s_Port** (WORD, INT, UINT)

Specification of slot number (high byte) and port number (low byte) of the MCU to which the data is transmitted.

- 16#xx01: COM1 on MCU in slot 16#xx
- 16#xx02: COM2 on MCU in slot 16#xx

Output

**d1_Para** (**MCU_PARA_DUT**)

Communication parameters defined in the predefined DUT

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the MCU unit does not exist in the specified slot or zero bytes should be sent.
- if the specified communication port does not exist

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the MCU unit does not exist in the specified slot or zero bytes should be sent.
- if the specified communication port does not exist

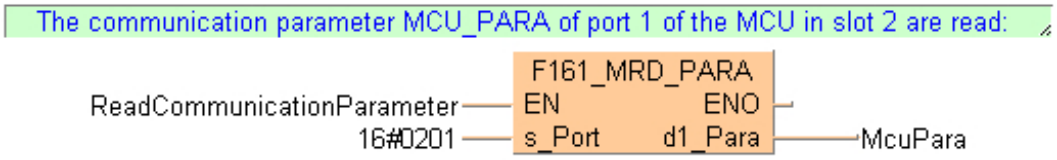
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | ReadCommunicationParameter | BOOL | FALSE |
| 1 | VAR | McuPara | MCU_PARA_DUT | |

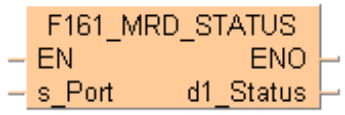
### LD body



# F161_MRD_STATUS

Get status data in RUN mode

Status data is read from the specified COM port of a Multi-Communication Unit.




## Parameters

Input

**s_Port** (WORD, INT, UINT)

Specification of slot number (high byte) and port number (low byte) of the MCU to which the data is transmitted.

- 16#xx01: COM1 on MCU in slot 16#xx
- 16#xx02: COM2 on MCU in slot 16#xx

Output

**d1_Status** (**MCU_STATUS_DUT**)

Communication parameters defined in the predefined DUT

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the MCU unit does not exist in the specified slot or zero bytes should be sent.
- if the specified communication port does not exist

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the MCU unit does not exist in the specified slot or zero bytes should be sent.
- if the specified communication port does not exist

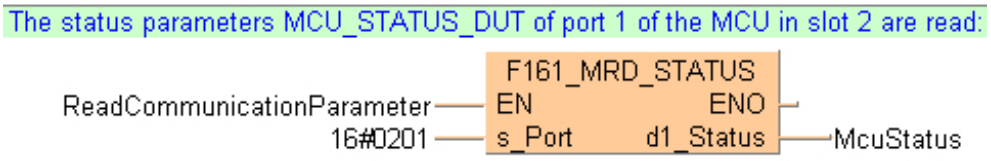
### Example

#### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | ReadCommunicationParameter | BOOL | FALSE |
| 1 | VAR | McuStatus | MCU_STATUS_DUT | |

#### LD body




## 10.3 PLC Link mode

PLC Link is an economic way of linking PLCs using a twisted-pair cable and the MEWNET protocol. Data is shared with all PLCs by means of dedicated internal flags called link flags (L) and data registers called link registers (LD). The statuses of the link flags and link registers of one PLC are automatically fed back to the other PLCs on the same network. The link flags and link registers of the PLCs contain areas for sending and areas for receiving data. Station numbers and link areas are allocated using the system registers.

Tip

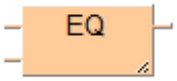
- For detailed information on setting the communication parameters and the link area, please refer to the hardware manuals of the corresponding units.
- Please refer to the corresponding hardware manual for further details.

# 11 Comparison instructions

# EQ

Equal to

The content of the accumulator is compared with the operand defined in the operand field. If both values are equal, "TRUE" is stored in the accumulator, otherwise "FALSE".




## Parameters

Input

**Unnamed input** (ANY)

1st input: value for comparison

**Unnamed input** (ANY)

2nd input: reference value

Output

**Unnamed output** (BOOL)

Result, TRUE if value for comparison is equal to the reference value

## Remarks

- Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.
- This function can be expanded to a maximum of 28 input contacts, see also modifying elements.
- When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is equal to the second value AND the second value is equal to the third value etc., TRUE will be written into result, otherwise FALSE.

# Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

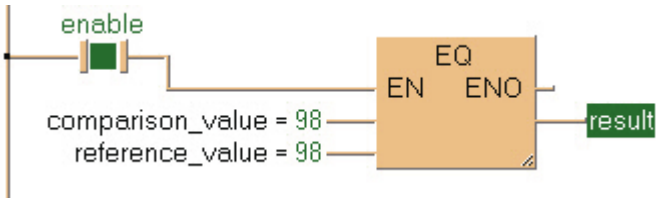
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | enable | BOOL | FALSE |
| 1 | VAR | comparison_value | INT | 0 |
| 2 | VAR | reference_value | INT | 0 |
| 3 | VAR | result | BOOL | FALSE |

In this example the input variables (**comparison_value**, **reference_value** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

### POU body

If **enable** is set to TRUE, the variable **comparison_value** is compared with the variable **reference_value**. If the values of the two variables are identical, the value TRUE will be written into result, otherwise FALSE.

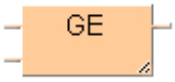
### LD body



# GE

Greater than or equal to

The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is greater or equal to the reference value, "TRUE" is stored in the accumulator, otherwise "FALSE".




## Parameters

Input

**Unnamed input** (ANY)

1st input: value for comparison

**Unnamed input** (ANY)

2nd input: reference value

Output

**Unnamed output** (BOOL)

Result, TRUE if value for comparison is greater than or equal to the reference value

## Remarks

- Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.
- This function can be expanded to a maximum of 28 input contacts, see also modifying elements.
- If the first value is greater than or equal to the second value AND the second value is greater than or equal to the third value etc., TRUE will be written into result, otherwise FALSE.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

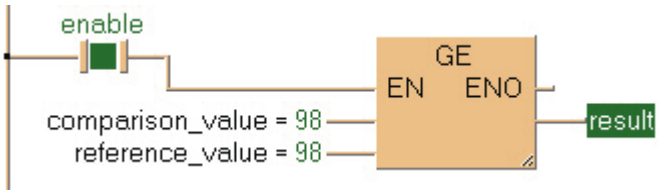
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | enable | BOOL | FALSE |
| 1 | VAR | comparison_value | INT | 0 |
| 2 | VAR | reference_value | INT | 0 |
| 3 | VAR | result | BOOL | FALSE |

In this example the input variables **comparison_value**, **reference_value** and **enable** have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

### POU body

If **enable** is set to TRUE, the **comparison_value** is compared with the **reference_value**. If the **comparison_value** is greater than or equal to the **reference_value**, the value TRUE will be written into **result**, otherwise FALSE.

### LD body



# GT

Greater than

The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is greater than the reference value, "TRUE" is stored in the accumulator, otherwise "FALSE".


## Parameters

Input

**Unnamed input** (ANY)

1st input: value for comparison

**Unnamed input** (ANY)

2nd input: reference value

Output

**Unnamed output** (BOOL)

Result, TRUE if value for comparison is greater than the reference value

## Remarks

- Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.
- This function can be expanded to a maximum of 28 input contacts, see also modifying elements.
- When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is greater than the second value AND the second value greater than third etc., TRUE will be written into result, otherwise FALSE.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

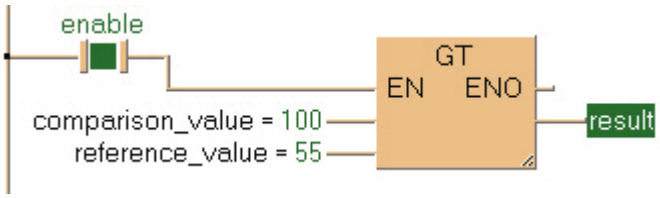
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | enable | BOOL | FALSE |
| 1 | VAR | comparison_value | INT | 0 |
| 2 | VAR | reference_value | INT | 0 |
| 3 | VAR | result | BOOL | FALSE |

In this example the input variables **comparison_value**, **reference_value** and **enable** have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

### POU body

If **enable** is set (TRUE), the **comparison_value** is compared with the **reference_value**. If the **comparison_value** is greater than the **reference_value**, the value TRUE will be written into **result**, otherwise FALSE.

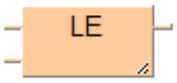
### LD body



# LE

Less than or equal to

The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is less or equal to the reference value, "TRUE" is stored in the accumulator, otherwise "FALSE".




## Parameters

Input

**Unnamed input** (ANY)

1st input: value for comparison

**Unnamed input** (ANY)

2nd input: reference value

Output

**Unnamed output** (BOOL)

Result, TRUE if value for comparison is less than or equal to the reference value

## Remarks

- Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.
- This function can be expanded to a maximum of 28 input contacts, see also modifying elements.
- When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is less than or equal to the second value AND the second value is less than or equal to the third value etc., TRUE will be written into result, otherwise FALSE.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

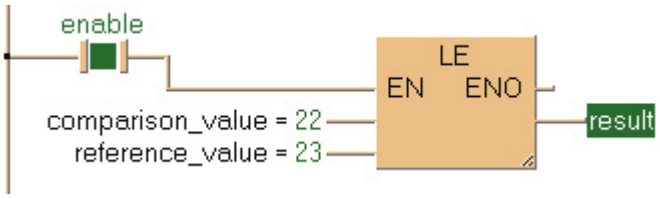
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | enable | BOOL | FALSE |
| 1 | VAR | comparison_value | INT | 0 |
| 2 | VAR | reference_value | INT | 0 |
| 3 | VAR | result | BOOL | FALSE |

In this example the input variables **comparison_value**, **reference_value** and **enable** have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

### POU body

If **enable** is set (TRUE), the **comparison_value** is compared with the variable **reference_value**. If the **comparison_value** is less than or equal to the **reference_value**, TRUE will be written into **result**, otherwise FALSE.

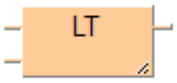
### LD body



# LT

Less than

The content of the accumulator is compared with the operand defined in the operand field. If the accumulator is less than the reference value, "TRUE" is stored in the accumulator, otherwise "FALSE".




## Parameters

Input

**Unnamed input** (ANY)

1st input: value for comparison

**Unnamed input** (ANY)

2nd input: reference value

Output

**Unnamed output** (BOOL)

Result, TRUE if value for comparison is less than the reference value

## Remarks

- Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.
- This function can be expanded to a maximum of 28 input contacts, see also modifying elements.
- When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is less than the second value AND the second value is less than the third value etc., TRUE will be written into result, otherwise FALSE.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

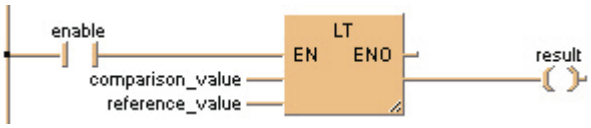
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | enable | BOOL | FALSE |
| 1 | VAR | comparison_value | INT | 0 |
| 2 | VAR | reference_value | INT | 0 |
| 3 | VAR | result | BOOL | FALSE |

In this example the input variables (**comparison_value**, **reference_value** and **enable**) have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

### POU body

If **enable** is set (TRUE), the **comparison_value** is compared with the **reference_value**. If the **comparison_value** is less than the **reference_value**, TRUE will be written into result, otherwise FALSE.

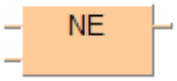
### LD body



# NE

Not equal

The content of the accumulator is compared with the operand defined in the operand field. If both values are not equal, "TRUE" is stored in the accumulator, otherwise "FALSE".




## Parameters

Input

**Unnamed input** (ANY)

1st input: value for comparison

**Unnamed input** (ANY)

2nd input: reference value

Output

**Unnamed output** (BOOL)

Result, TRUE if value for comparison is unequal to the reference value, otherwise FALSE

## Remarks

- Inputs can be of any data type; all input variables must be of the same data type though. Output must be of type BOOL.
- This function can be expanded to a maximum of 28 input contacts, see also modifying elements.
- When using more inputs, the first input is compared with the second, the second input is compared with the third input etc. If the first value is not equal to the second value AND the second value is not equal to the third value etc., TRUE will be written into result, otherwise FALSE.

# Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

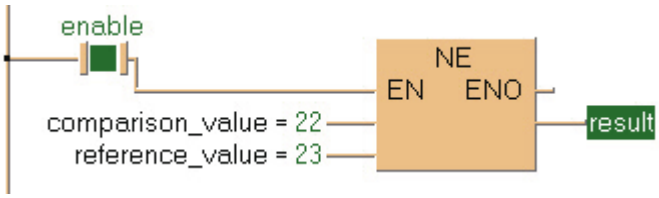
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | enable | BOOL | FALSE |
| 1 | VAR | comparison_value | INT | 0 |
| 2 | VAR | reference_value | INT | 0 |
| 3 | VAR | result | BOOL | FALSE |

In this example the input variables **comparison_value**, **reference_value** and **enable** have been declared. Instead, you may enter constants directly into the function (enable input e.g. for tests).

### POU body

If **enable** is set (TRUE), the **comparison_value** is compared with the **reference_value**. If the two values are unequal, TRUE will be written into **result**, otherwise FALSE.

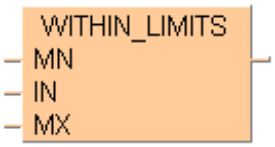
### LD body



# WITHIN_LIMITS

Evaluate if a value is inside the limit

This instruction evaluates whether the value at the input **IN** is within the limits set at minimum **MN** and maximum **MX**.




## Parameters

Input

**MN** (ANY)

Minimum limit

**IN** (ANY)

Value compared to the limits

**MX** (ANY)

Maximum limit

Output

**VAR_OUT** (BOOL)

TRUE if the input value at **IN** falls within the lower and upper limits

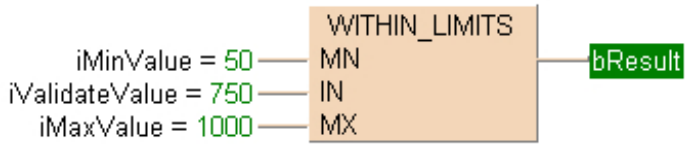
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iMinValue | INT | 50 |
| 1 | VAR | iValidateValue | INT | 750 |
| 2 | VAR | iMaxValue | INT | 1000 |
| 3 | VAR | bResult | BOOL | FALSE |

### LD body


WITHIN_LIMITS
iMinValue = 50
MN
iValidateValue = 750
IN
iMaxValue = 1000
MX
bResult

## 11.8 FP instructions

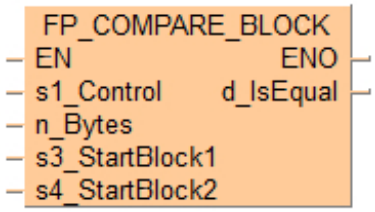
Tip

Advantages of FP instructions

# FP_COMPARE_BLOCK

Comparison of data blocks

Compares the contents of the data block specified by **s3_StartBlock1** with the contents of the data block specified by **s4_StartBlock2** if the trigger **EN** is TRUE. **s1_Control** specifies the byte positions and **n_Bytes** the number of bytes.




## Parameters

Input

**s1_Control** (WORD, INT, UINT)

Control code specifying byte positions

**n_Bytes** (WORD, INT, UINT)

Number of bytes to be compared. Values: 1–99

**s3_StartBlock1** (WORD, INT, UINT)

Starting address of data block 1 to be compared to **s4_StartBlock2**

**s4_StartBlock2** (WORD, INT, UINT)

Starting address of data block 2 to be compared to **s3_StartBlock1**

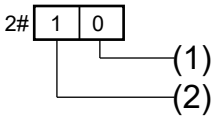
Output

**d_IsEqual** (BOOL)

Compare operation result
TRUE if **s3_StartBlock1** = **s4_StartBlock2**

## Remarks

- Specifying the control code **s1_Control**




(1) Starting byte position of the data block specified by **s3_StartBlock1**
1: starting from higher byte
0: starting from lower byte

(2) Starting byte position of the data block specified by **s4_StartBlock2**
1: starting from higher byte
0: starting from lower byte

- The variables **s3_StartBlock1** and **s4_StartBlock2** have to be of the same data type.

## Example

### POU header

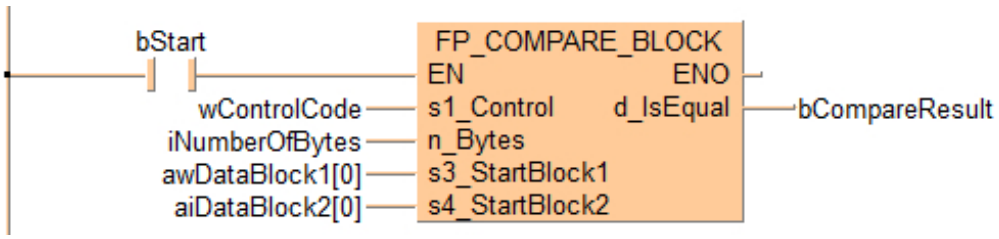
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Klasse | Bezeichner | Typ | Initial | Kommentar |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | wControlCode | WORD | 2#11 | s3 starting from upper byte |
| 2 | VAR | iNumberOfBytes | INT | 6 | number of bytes to be compared |
| 3 | VAR | awDataBlock1 | ARRAY [0..2] OF WORD | [3(0)] | |
| 4 | VAR | aiDataBlock2 | ARRAY [0..5] OF INT | [6(1234)] | |
| 5 | VAR | bCompareResult | BOOL | FALSE | |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

### LD body



## 11.9 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

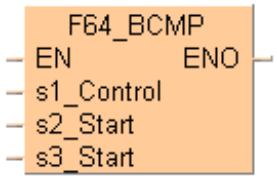
Advantages of IEC instructions

Advantages of FP instructions

# F64_BCMP

Comparison of data blocks

Compares the contents of data block specified by **s2_Start** with the contents of data block specified by **s3_Start** according to the contents specified by **s1_Control** if the trigger **EN** is in the ON-state.




## Parameters

Input

**s1_Control** (WORD)

control code specifying byte positions and number of bytes to be compared

**s2_Start** (WORD, INT, UINT)

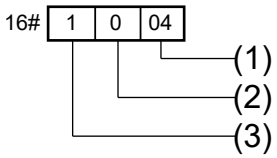
starting 16-bit area to be compared to **s3_Start**

**s3_Start** (WORD, INT, UINT)

starting 16-bit area to be compared to **s2_Start**

## Remarks

Specifying the control code **s1_Control**




(1) Number of bytes to be compared
range: 16#01–16#99 (BCD)

(2) Starting byte position of data block specified by **s2_Start**
1: starting from higher byte
0: starting from lower byte

(3) Starting byte position of data block specified by **s3_Start**
1: starting from higher byte
0: starting from lower byte

The compare operation result is stored in the special internal flag R900B. When **s2_Start=s3_Start**, the special internal flag is in the ON-state.

The flag **sys_bIsEqual** used for the compare instruction is renewed each time a compare instruction is executed. Therefore the program that uses R900B should be just after **F64_BCMP** .

The variables **s1** and **s2** have to be of the same data type.

### Example

#### POU header

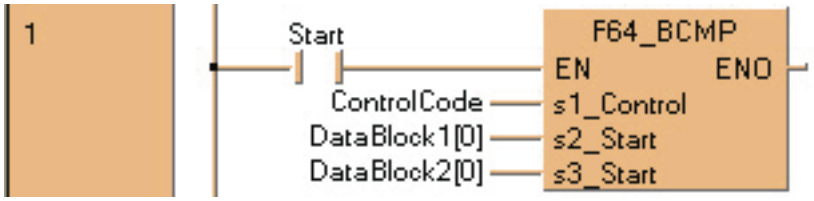
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | ControlCode | WORD | 16#1106 | s2 starting from upper byte |
| 2 | VAR | DataBlock1 | ARRAY [0..5] OF INT | [6(1234)] | s3 starting from upper byte |
| 3 | VAR | DataBlock2 | ARRAY [0..5] OF INT | [6(1234)] | compare 6 bytes |
| 4 | VAR | equal_block | BOOL | FALSE | |

#### POU body

When the variable **start** is set to TRUE, the function is carried out.

#### LD body




## 11.9.2 Further comparison instructions

If you need information on one of the following comparison instructions, please refer to the corresponding standard operators:

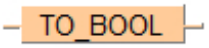
| | | | | | |
|---|---|---|---|---|---|
| ST= | AN= | OR= | STD= | AND= | ORD= |
| ST<> | AN<> | OR<> | STD<> | AND<> | ORD<> |
| ST> | AN> | OR> | STD> | AND> | ORD> |
| ST>= | AN>= | OR>= | STD>= | AND>= | ORD>= |
| ST< | AN< | OR< | STD< | AND< | ORD< |
| ST<= | AN<= | OR<= | STD<= | AND<= | ORD<= |

# 12 Conversion instructions

# TO_BOOL

Overloaded conversion to BOOL

This instruction converts a value of any allowed data type into a value of the data type BOOL.




## Parameters

Input

**Unnamed input** (INT, UINT, DINT, UDINT, REAL, LREAL, BOOL, WORD, DWORD)

Value to be converted

Output

**Unnamed output** (BOOL)

Conversion result

## Example

### POU header

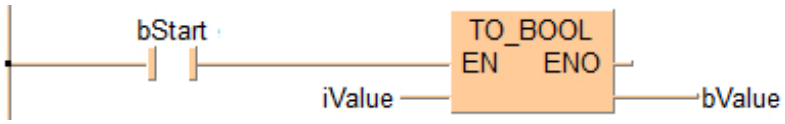
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bValue | BOOL | FALSE |
| 1 | VAR | iValue | INT | 0 |
| 2 | VAR | bStart | BOOL | FALSE |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

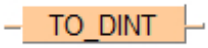
### LD body



# TO_DINT

Overloaded conversion to DOUBLE INTEGER

This instruction converts a value of any allowed data type into a value of the data type DINT.




## Parameters

Input

**Unnamed input** (INT, UINT, DINT, UDINT, REAL, LREAL, BOOL, WORD, DWORD)

Value to be converted

Output

**Unnamed output** (DINT)

Conversion result

## Example

### POU header

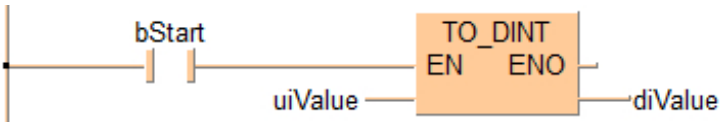
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | diValue | DINT | 0 |
| 1 | VAR | uiValue | UINT | 0 |
| 2 | VAR | bStart | BOOL | FALSE |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

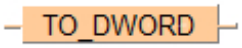
### LD body



# TO_DWORD

Overloaded conversion to DOUBLE WORD

This instruction converts a value of any allowed data type into a value of the data type DWORD.




## Parameters

Input

**Unnamed input** (INT, UINT, DINT, UDINT, REAL, LREAL, BOOL, WORD, DWORD)

Value to be converted

Output

**Unnamed output** (DWORD)

Conversion result

## Example

### POU header

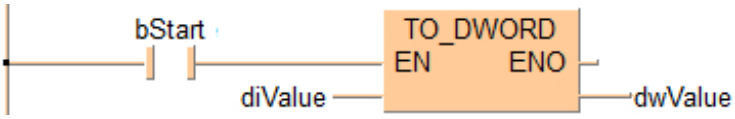
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | dwValue | DWORD | 0 |
| 1 | VAR | diValue | DINT | 0 |
| 2 | VAR | bStart | BOOL | FALSE |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

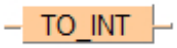
### LD body



# TO_INT

Overloaded conversion to INTEGER

This instruction converts a value of any allowed data type into a value of the data type INT.




## Parameters

Input

**Unnamed input** (INT, UINT, DINT, UDINT, REAL, LREAL, BOOL, WORD, DWORD)

Value to be converted

Output

**Unnamed output** (INT)

Conversion result

## Example

### POU header

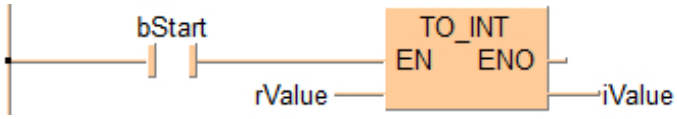
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iValue | INT | 0 |
| 1 | VAR | bStart | BOOL | FALSE |
| 2 | VAR | rValue | REAL | 0.0 |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

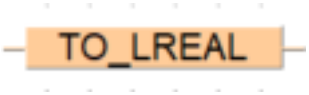
### LD body



# TO_LREAL

Overloaded conversion to LREAL

This instruction converts a value of any allowed data type into a value of the data type LREAL.




**Unnamed input** (INT, UINT, DINT, UDINT, REAL, LREAL, BOOL, WORD, DWORD)

Value to be converted

Output

**Unnamed output** (LREAL)

Conversion result

## Example

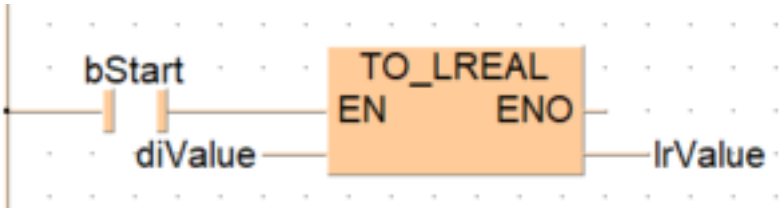
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | lrValue | LREAL | 0.0 |
| 2 | VAR | diValue | DINT | 0 |
| 3 | VAR | bStart | BOOL | FALSE |

### LD body

When the variable **bStart** is set to TRUE, the function is carried out.



# TO_REAL

Overloaded conversion to REAL

This instruction converts a value of any allowed data type into a value of the data type REAL.


## Parameters

Input

**Unnamed input** (INT, UINT, DINT, UDINT, REAL, LREAL, BOOL, WORD, DWORD)

Value to be converted

Output

**Unnamed output** (REAL)

Conversion result

## Example

### POU header

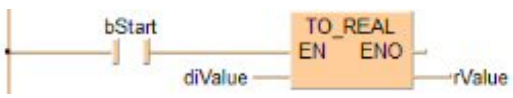
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | rValue | REAL | 0.0 |
| 1 | VAR | diValue | DINT | 0 |
| 2 | VAR | bStart | BOOL | FALSE |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

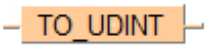
### LD body



# TO_UDINT

Overloaded conversion to unsigned DOUBLE INTEGER

This instruction converts a value of any allowed data type into a value of the data type UDINT.




## Parameters

Input

**Unnamed input** (INT, UINT, DINT, UDINT, REAL, LREAL, BOOL, WORD, DWORD)

Value to be converted

Output

**Unnamed output** (UDINT)

Conversion result

## Example

### POU header

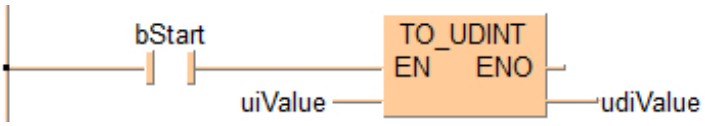
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | udiValue | UDINT | 0 |
| 1 | VAR | uiValue | UINT | 0 |
| 2 | VAR | bStart | BOOL | FALSE |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

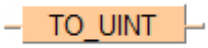
### LD body



# TO_UINT

Overloaded conversion to unsigned INTEGER

This instruction converts a value of any allowed data type into a value of the data type UINT.




## Parameters

Input

**Unnamed input** (INT, UINT, DINT, UDINT, REAL, LREAL, BOOL, WORD, DWORD)

Value to be converted

Output

**Unnamed output** (UINT)

Conversion result

## Example

### POU header

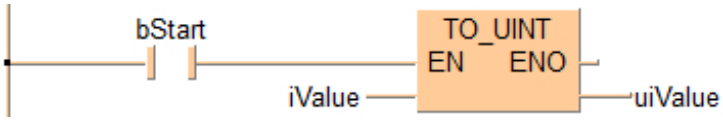
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | uiValue | UINT | 0 |
| 1 | VAR | iValue | INT | 0 |
| 2 | VAR | bStart | BOOL | FALSE |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

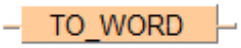
### LD body



# TO_WORD

Overloaded conversion to WORD

This instruction converts a value of any allowed data type into a value of the data type WORD.




## Parameters

Input

**Unnamed input** (INT, UINT, DINT, UDINT, REAL, LREAL, BOOL, WORD, DWORD)

Value to be converted

Output

**Unnamed output** (WORD)

Conversion result

## Example

### POU header

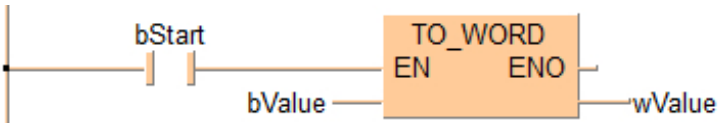
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | wValue | WORD | 0 |
| 1 | VAR | bValue | BOOL | FALSE |
| 2 | VAR | bStart | BOOL | FALSE |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

### LD body



## 12.10 Conversion to WORD

# BOOL16_TO_WORD

BOOL16 into WORD

This function copies a variable of the special data type BOOL16 (an array with 16 elements of the data type BOOL or a DUT of 16 members of the data type BOOL) at the input into the data type WORD at the output.


## Parameters

Input

**Unnamed input**ARRAY OF BOOL

ARRAY with 16 elements

Output

**Unnamed output** (WORD)

Conversion result

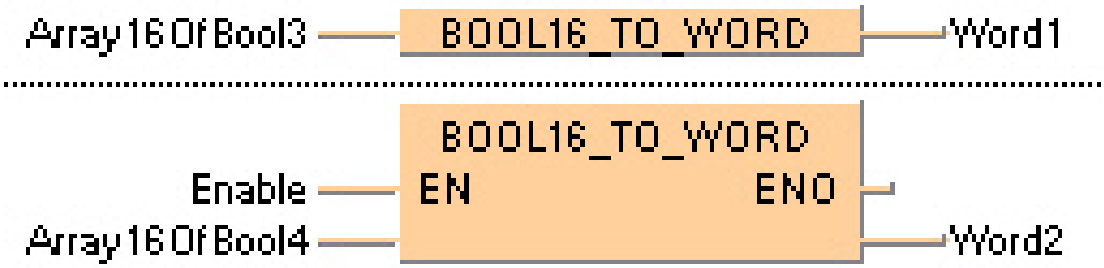
## Remarks

If the required data type is supported, we recommend using the overloaded instruction **TO_WORD** (page 658)

### POU header

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Enable | BOOL | FALSE |
| 1 | VAR | Word_1 | WORD | 0 |
| 2 | VAR | Word_2 | WORD | 0 |
| 3 | VAR | Array16OfBool1 | ARRAY [0..15] OF BOOL | [16(FALSE)] |
| 4 | VAR | Array16OfBool2 | ARRAY [0..15] OF BOOL | [16(FALSE)] |

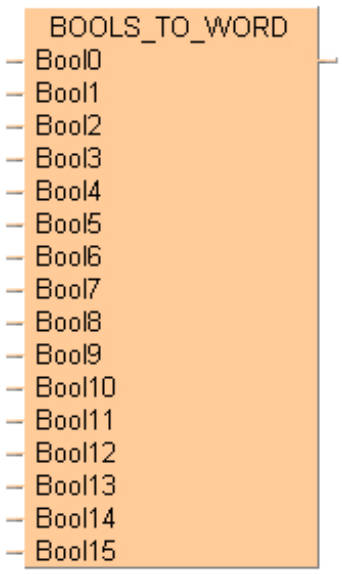
**Body with and without EN/ENO:**



# BOOLS_TO_WORD

16 variables of the data type BOOL into WORD

This function converts 16 values of the data type BOOL bit-wise into a value of the data type WORD.




## Parameters

Input

**Bool0**...**Bool15** (BOOL)

16 input variables of the data type BOOL

Output

**VAR_OUT** (WORD)

Conversion result

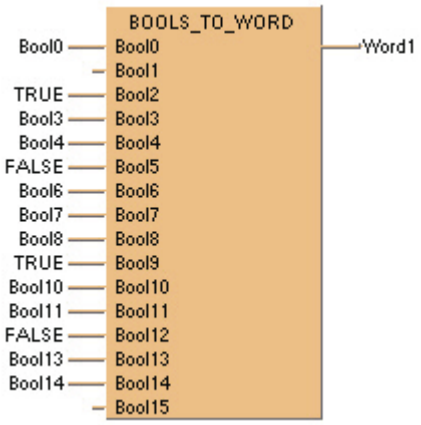
## Remarks

- If the required data type is supported, we recommend using the overloaded instruction**TO_WORD**
- The inputs **Bool0** to **Bool15** need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Such unused inputs are assumed to be FALSE. No program code is generated for these inputs (or for any input allocated with the constants TRUE or FALSE). Program code is only generated for inputs to which a variable is allocated.

### POU header

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Word0 | WORD | 0 |
| 1 | VAR | Bool0 | BOOL | FALSE |
| 2 | VAR | Bool1 | BOOL | FALSE |
| 3 | VAR | Bool2 | BOOL | FALSE |
| 4 | VAR | Bool3 | BOOL | FALSE |
| 5 | VAR | Bool4 | BOOL | FALSE |
| 6 | VAR | Bool5 | BOOL | FALSE |
| 7 | VAR | Bool6 | BOOL | FALSE |
| 8 | VAR | Bool7 | BOOL | FALSE |
| 9 | VAR | Bool8 | BOOL | FALSE |
| 10 | VAR | Bool9 | BOOL | FALSE |
| 11 | VAR | Bool10 | BOOL | FALSE |
| 12 | VAR | Bool11 | BOOL | FALSE |
| 13 | VAR | Bool12 | BOOL | FALSE |
| 14 | VAR | Bool13 | BOOL | FALSE |
| 15 | VAR | Bool14 | BOOL | FALSE |
| 16 | VAR | Bool15 | BOOL | FALSE |

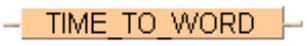
### LD body



# TIME_TO_WORD

TIME into WORD

**TIME_TO_WORD** converts a value of the data type TIME into a value of the data type WORD.




## Parameters

Input

**Unnamed input** (TIME)

input data type

Output

**Unnamed output** (WORD)

Conversion result

## Remarks

If the required data type is supported, we recommend using the overloaded instruction **TO_WORD**

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | time_value | TIME | T#0s |
| 1 | VAR | WORD_value | WORD | 0 |

This example uses variables. You can also use a constant for the input variable.

## POU body

**time_value** of the data type TIME is converted into a value of the data type WORD. The result will be written into the output variable **WORD_value**.

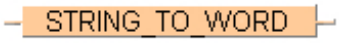
### LD body

# STRING_TO_WORD

STRING (hexadecimal format) into WORD

This function converts a STRING in hexadecimal format into a value of the data type WORD.

Thereby the attached string is first converted into a value of the data type STRING[32]. Finally this is converted into a value of the data type WORD via a sub-program of approx. 270 steps that is also used in the functions **STRING_TO_INT**, **STRING_TO_WORD**, **STRING_TO_DINT** and **STRING_TO_DWORD**.




Input

**Unnamed input** (STRING)

input data type

Output

**Unnamed output** (WORD)

Conversion result

## Remarks

If the required data type is supported, we recommend using the overloaded instruction TO_WORD

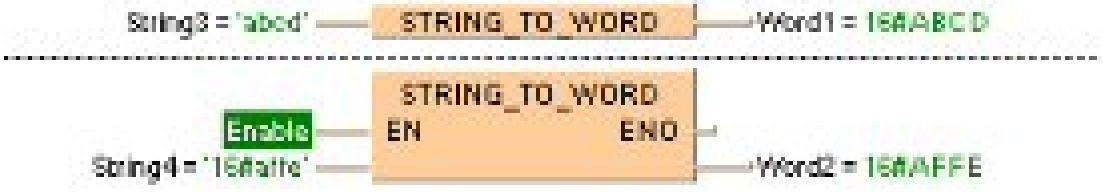
Permissible format: '[Space][Hexadecimal numbers][Space]' e.g. ' afFE '

Permissible characters:

- Space ␣
- Sign "+" (plus), "-" (minus)
- Hexadecimal numbers in the ranges "0 - 9", "A - F" or "a - f".

The analysis ends with the first non-hexadecimal number.

### Example with and without EN/ENO:



# STRING_TO_WORD_STEPSAVER

STRING (hexadecimal format right-justified) into WORD

This function converts the string with the maximum possible number of characters that are right aligned in hexadecimal format into a value of the data type WORD.


## Parameters

Input

**Unnamed input** (STRING)

input data type

Output

**Unnamed output** (WORD)

Conversion result

## Remarks

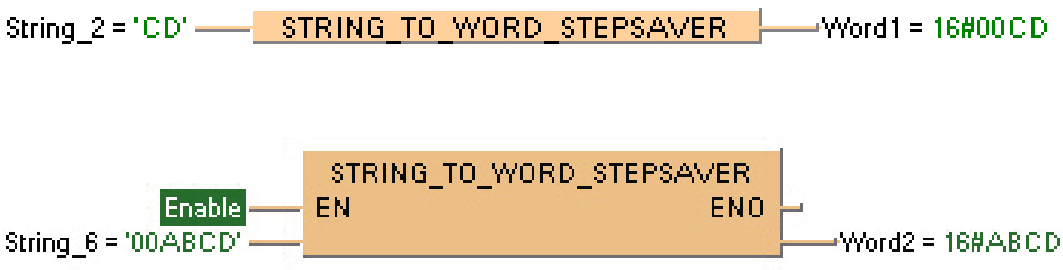
The basic instruction F72_A2HEX is used. The PLC delivers an operation error especially when a character appears that is not a hexadecimal number “0 - 9" or "A-F”.

Acceptable Format for STRING[4]: 'Hex1Hex2Hex3Hex4' e.g. perhaps 'AFFE'

Acceptable characters:

- Hex1 to Hex4: Hexadecimal numbers in the range "0 - 9“ or "A - F“ (not "a - f").

### Example with and without EN/ENO:



### Conversion examples:

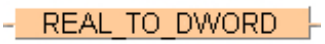
| Input | Defined as | Results in |
|---|---|---|
| 'D' | STRING[1] | 16#D |
| 'CD' | STRING[2] | 16#CD |
| 'BCD' | STRING[3] | 16#BCD |
| 'ABCD' | STRING[4] | 16#ABCD |
| '0ABCD' | STRING[5] | 16#ABCD |
| '00ABCD' | STRING[6] | 16#ABCD |

## 12.11 Conversion to DWORD

# REAL_TO_DWORD

REAL into DOUBLE WORD

**REAL_TO_DWORD** moves bit set information of a REAL variable to a DWORD variable. The same functionality can be obtained using **DWORD_OVERLAPPING_DUT**.




## Parameters

Input

**Unnamed input** (REAL)

Input data type

Output

**Unnamed output** (DWORD)

Conversion result

## Remarks

If the required data type is supported, we recommend using the overloaded instruction **TO_DWORD**

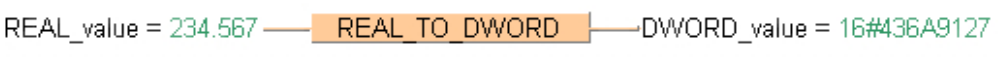
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | REAL_value | REAL | 234.567 |
| 1 | VAR | DWORD_value | DWORD | 16#00000000 |

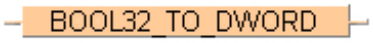
### LD body



# BOOL32_TO_DWORD

BOOL32 into DOUBLE WORD

This function copies a variable of the special data type BOOL32 (an array with 32 elements of the data type BOOL or a DUT of 32 members of the data type BOOL) at the input into the data type DWORD at the output.




## Parameters

Input

**Unnamed input**ARRAY OF BOOL

ARRAY with 32 elements

Output

**Unnamed output** (DWORD)

Conversion result

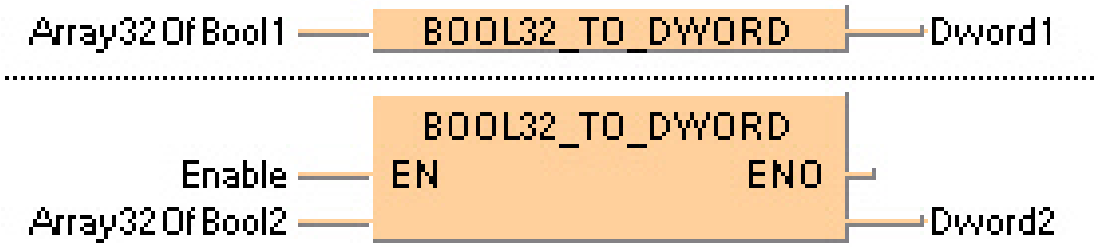
## Remarks

If the required data type is supported, we recommend using the overloaded instruction TO_DWORD

### POU header

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Enable | BOOL | FALSE |
| 1 | VAR | Array32OfBool1 | ARRAY [0..31] OF BOOL | [32(FALSE)] |
| 2 | VAR | Array32OfBool2 | ARRAY [0..31] OF BOOL | [32(FALSE)] |
| 3 | VAR | DWord1 | DWORD | 0 |
| 4 | VAR | DWord2 | DWORD | 0 |

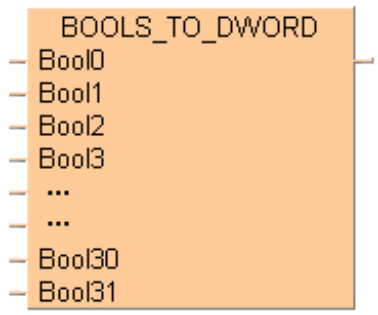
**Body with and without EN/ENO:**



# BOOLS_TO_DWORD

32 variables of the data type BOOL into DWORD

This function converts 32 values of the data type BOOL bit-wise into a value of the data type DWORD.




## Parameters

Input

**Bool0**...**Bool31** (BOOL)

32 input variables of the data type BOOL

Output

**VAR_OUT** (DWORD)

Conversion result

## Remarks

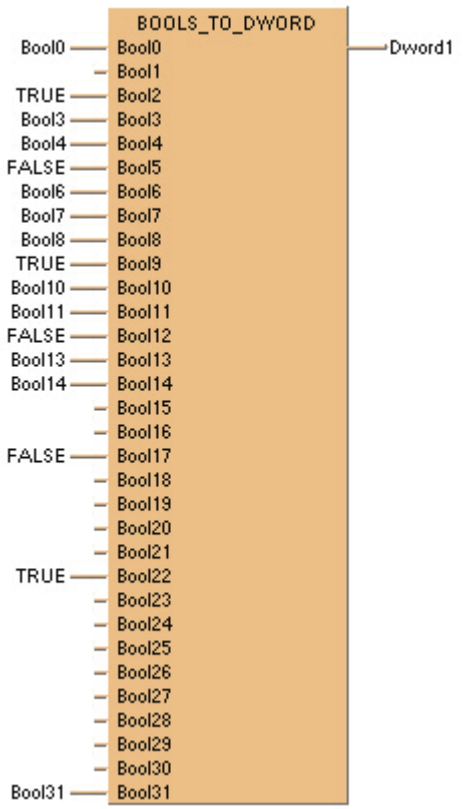
- If the required data type is supported, we recommend using the overloaded instruction **TO_DWORD**
- The inputs **Bool0** to **Bool31** need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Such unused inputs are assumed to be FALSE. No program code is generated for these inputs (or for any input allocated with the constants TRUE or FALSE). Program code is only generated for inputs to which a variable is allocated.

### POU header

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | dWord1 | DWORD | 0 |
| 1 | VAR | Bool0 | BOOL | FALSE |
| 2 | VAR | Bool1 | BOOL | FALSE |
| 3 | VAR | Bool2 | BOOL | FALSE |
| 4 | VAR | Bool3 | BOOL | FALSE |
| 5 | VAR | Bool4 | BOOL | FALSE |
| 6 | VAR | Bool5 | BOOL | FALSE |
| 7 | VAR | Bool6 | BOOL | FALSE |
| 8 | VAR | Bool7 | BOOL | FALSE |

etc. to Bool31

### Body with and without EN/ENO:



# TIME_TO_DWORD

TIME into DOUBLE WORD

**TIME_TO_DWORD** converts a value of the data type TIME into a value of the data type DWORD. The time 10ms corresponds to the value 1, e.g. an input value of T#1s is converted to the value 100 (16#64).


## Parameters

Input

**Unnamed input** (TIME)

Input data type

Output

**Unnamed output** (DWORD)

Conversion result

## Remarks

If the required data type is supported, we recommend using the overloaded instruction TO_DWORD

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

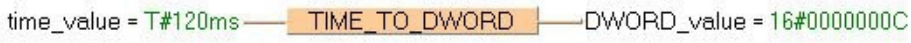
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | time_value | TIME | T#120ms | |
| 1 | VAR | DWORD_value | DWORD | 0 | result: 16#C |

This example uses variables. You can also use a constant for the input variable.

## POU body

**time_value** of the data type TIME is converted to value of the data type DWORD and written into the output variable **DWORD_value**.

### LD body

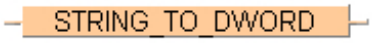


# STRING_TO_DWORD

STRING (hexadecimal format) into DOUBLE WORD

This function converts a string in hexadecimal formal into a value of the data type DWORD.

At first the string is converted into a value of the data type STRING[32]. Finally this is converted into a value of the data type DWORD in a subprogram of approximately 270 steps, which is also used by the functions **STRING_TO_INT**, **STRING_TO_WORD**, **STRING_TO_DINT** and **STRING_TO_DWORD**.




## Parameters

Input

**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (DWORD)

Conversion result

## Remarks

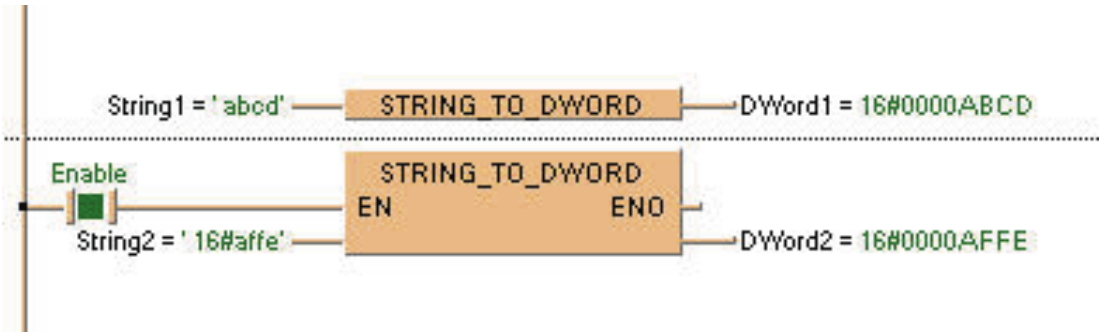
Acceptable Format: _'[Space][Hexadecimal number][Space]' e.g. perhaps ' **afFE** '

Acceptable characters:

- Space ␣
- Sign "+" (plus), "-" (minus)
- Hexadecimal numbers in the ranges "0 - 9", "A - F" or "a - f".

The analysis ends with the first non-decimal number.

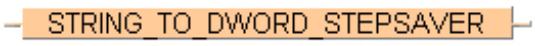
**Example with and without EN/ENO:**



# STRING_TO_DWORD_STEPSAVER

STRING (hexadecimal format right-justified) into DOUBLE WORD

This function converts the string with the maximum possible number of characters that are right aligned in hexadecimal format into a value of the data type DWORD.




## Parameters

Input

**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (DWORD)

Conversion result

## Remarks

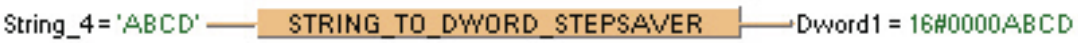
Acceptable Format for STRING[8]:

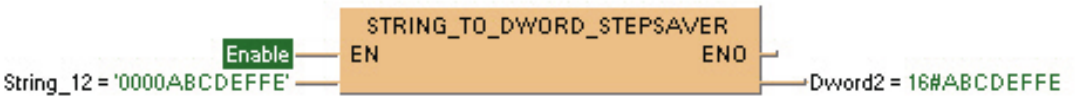
'Hex1Hex2Hex3Hex4Hex5Hex6Hex7Hex8' e.g. '001AAFFE'

Acceptable characters:

Hex1–Hex8: Hexadecimal numbers in the range "0–9“ or "A–F“ (not "a–f").

### Example




### Conversion examples:

| Input | Defined as | Results in |
|---|---|---|
| 'FE' | STRING[2] | 16#FE |
| 'EFFE' | STRING[4] | 16#EFFE |

| Input | Defined as | Results in |
|---|---|---|
| 'CDEFFE' | STRING[6] | 16#CDEFFE |
| 'ABCDEFFE' | STRING[8] | 16#ABCDEFFE |
| '00ABCDEFFE' | STRING[10] | 16#ABCDEFFE |

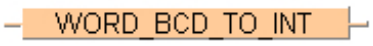
The basic instruction F72_A2HEX is used. The PLC delivers an operation error especially when a character appears that is not a hexadecimal number “0 - 9" or "A - F”.

## 12.12 Conversion to INT

# WORD_BCD_TO_INT

BCD value of WORD into INTEGER

**WORD_BCD_TO_INT** converts a binary coded BCD value of WORD into binary values of type INT.




## Parameters

Input

**Unnamed input** (WORD_BCD)

Input data type

Output

**Unnamed output** (INT)

Conversion result

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | BCD_value_16bit | WORD | 0 |
| 1 | VAR | INT_value | INT | 0 |

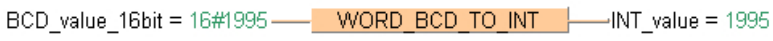
This example uses variables. You can also use a constant for the input variable.

BCD constants can be expressed in Control FPWIN Pro7 as follows: 2#0001100110010101 or 16#1995

## POU body

**BCD_value_16bit** of the data type WORD is converted into an INTEGER value. The converted value is written into output variable **INT_value**.

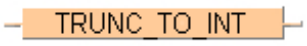
## LD body



# TRUNC_TO_INT

Truncate (cut off) decimal digits of REAL or LREAL input variable, convert to INTEGER

**TRUNC_TO_INT** cuts off the decimal digits of a REAL, LREAL number and delivers an output variable of the data type INTEGER.




## Parameters

Input

**Unnamed input** (REAL, LREAL)

Input data type

Output

**Unnamed output** (INT)

Conversion result

## Remarks

- If the required data type is supported, we recommend using the overloaded instruction **TO_INT**
- If the decimal digits are cut off, positive numbers will be decreased towards zero and negative numbers will be increased towards zero.
- The first 16 bits of the input variable are assigned to the output variable.

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | REAL_value | REAL | 0.0 | number betw. -32768.99 ... +32767 |
| 1 | VAR | INT_value | INT | 0 | number betw. -32768 ... +32768 |

This example uses variables. You can also use a constant for the input variable.

### POU body

The decimal digits of **REAL_value** are cut off. The **result** is stored as a 16-bit INTEGER in **INT_value**.

### LD body

# TIME_TO_INT

TIME into INTEGER

**TIME_TO_INT** converts a value of the data type TIME into a value of the data type INT.


## Parameters

Input

**Unnamed input** (TIME)

Input data type

Output

**Unnamed output** (INT)

Conversion result

## Remarks

If the required data type is supported, we recommend using the overloaded instruction TO_INT

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | tme_value | TIME | T#0s |
| 1 | VAR | INT_value | INT | 0 |

This example uses variables. You can also use a constant for the input variable.

## POU body

**time_value** of the data type TIME is converted into a value of the data type INTEGER. The result will be written into the output variable **INT_value**.

### LD body

# STRING_TO_INT

STRING (decimal format) into INTEGER

This function converts a STRING in decimal format into a value of the data type INT.


## Parameters

Input

**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (INT)

Conversion result

## Remarks

- If the required data type is supported, we recommend using the overloaded instruction TO_INT
- Thereby the attached string is first converted into a value of the data type STRING[32]. Finally this is converted into a value of the data type INT via a sub-programm of approx. 270 steps that is also used in the functions **STRING_TO_INT**, **STRING_TO_WORD**, **STRING_TO_DINT** and **STRING_TO_DWORD**.

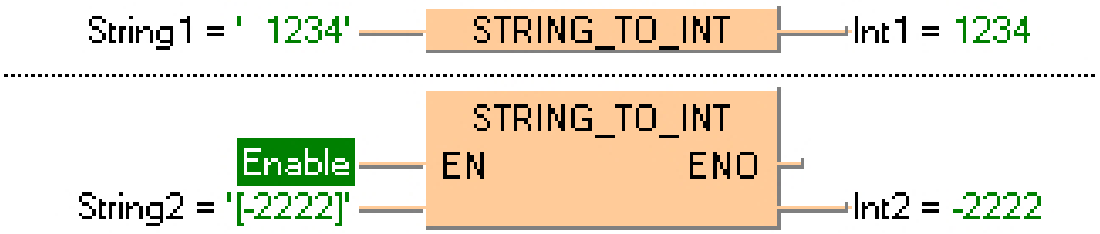
Permissible format: '[Space][Sign][Decimal numbers][Space]' e.g. ' 123456 '

Permissible characters:

- Space ␣
- Sign "+" (plus), "-" (minus)
- Decimal numbers "0 - 9"

The analysis ends with the first non-decimal number.

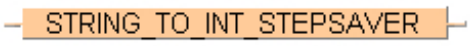
**Example**


String1 = ' 1234'
STRING_TO_INT
Int1 = 1234
STRING_TO_INT
Enable
EN
ENO
String2 = '[-2222]'
Int2 = -2222

# STRING_TO_INT_STEPSAVER

STRING (decimal format right-justified) into INTEGER

This function converts a right-justifed decimal number in a string into a value of the data type INT.




## Parameters

Input

**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (INT)

Conversion result

## Remarks

The basic instruction **F76_A2BIN** with approx. 7 steps is used. The PLC delivers an operation error especially when a character appears that is not a decimal number “0 - 9”, not a “+” or “-“ or not a space.

Acceptable Format: '[Space][Sign][Decimal number]' e.g. '␣123456'

Acceptable characters:

- Space ␣
- Sign "+" (plus), "-" (minus)
- Decimal numbers "0 - 9"

The analysis ends with the first non-decimal number.

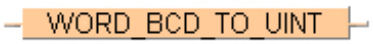
### Example

## 12.13 Conversion to UINT

# WORD_BCD_TO_UINT

Binary value of WORD into unsigned INTEGER

**WORD_BCD_TO_UINT** converts a binary coded value of the data type WORD into a value of the data type Unsigned INTEGER.




## Parameters

Input

**Unnamed input** (WORD_BCD)

input data type

Output

**Unnamed output** (UINT)

Conversion result

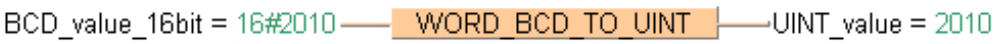
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UINT_value | UINT | 0 |
| 1 | VAR | BCD_value_16bit | WORD | 16#2010 |

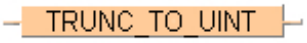
### LD body



# TRUNC_TO_UINT

Truncate (cut off) decimal digits of REAL or LREAL input variable, convert to Unsigned INTEGER

**TRUNC_TO_UINT** cuts off any digits following the decimal of a REAL, LREAL number and delivers an output variable of the data type Unsigned INTEGER.




## Parameters

Input

**Unnamed input** (REAL, LREAL)

input data type

Output

**Unnamed output** (UINT)

Conversion result

## Remarks

If the decimal digits are cut off, positive numbers will be decreased towards zero and negative numbers will be increased towards zero.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if input variable does not have the data type REAL

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if output variable is greater than a 16-bit INTEGER

**sys_bIsCarry** (turns to TRUE for one scan)

- if output variable is zero

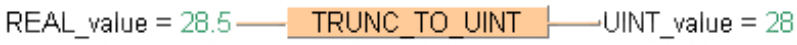
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UINT_value | UINT | 0 |
| 1 | VAR | REAL_value | REAL | 28.5 |

### LD body



# STRING_TO_UINT

STRING (decimal format) into unsigned INTEGER

**STRING_TO_UINT** converts a string in decimal formal into a value of the data type Unsigned INTEGER.

First, the string is converted into a value of the data type STRING[32], which is subsequently converted into a value of the data type UINT in a subprogram with approximately 270 steps. The subprogram is also used by the functions **STRING_TO_INT**, **STRING_TO_WORD**, **STRING_TO_UDINT** and **STRING_TO_DWORD**.


## Parameters

Input

**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (UINT)

Conversion result

## Remarks

Acceptable Format:

'[Space][Sign][Decimal number][Space]', e.g. ' 123456 '

Acceptable characters:

- Space
  Space “␣“
- Sign
  "+“ (plus), "-" (minus)
- Decimal Number
  Decimal numbers "0“ - "9“

The analysis ends with the first non-decimal number.

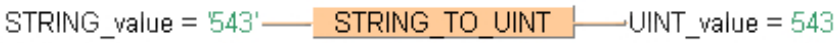
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UINT_value | UINT | 0 |
| 1 | VAR | STRING_value | STRING[8] | '543' |

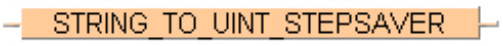
### LD body



# STRING_TO_UINT_STEPSAVER

STRING (decimal format right-justified) into unsigned INTEGER

**STRING_TO_UINT_STEPSAVER** converts a right-justifed decimal number in a string into a value of the data type Unsigned INTEGER.




## Parameters

Input

**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (UINT)

Conversion result

## Remarks

The basic instruction F76_A2BIN with approx. 7 steps is used. The PLC issues an operation error especially if anything other than acceptable characters are used (see the following table "Acceptable characters").

Acceptable Format: '[Space][Sign][Decimal number]', e.g. '␣123456'

Acceptable characters:

- Space
  Space “␣“
- Sign
  "+“ (plus), "-" (minus)
- Decimal Number
  Decimal numbers "0“ - "9“

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UINT_value | UINT | 0 |
| 1 | VAR | STRING_value | STRING[8] | '543' |

### LD body

## 12.14 Conversion to DINT

# DWORD_BCD_TO_DINT

Binary value of DWORD into DOUBLE INTEGER

**DWORD_BCD_TO_DINT** converts a binary coded value of the data type DWORD into a binary value of the data type DINT in order to be able to process a BCD value in double word format.


## Parameters

Input

**Unnamed input** (DWORD_BCD)

Input data type

Output

**Unnamed output** (DINT)

Conversion result

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DINT_value | DINT | 0 |
| 1 | VAR | BCD_value_32bit | DWORD | 0 |

This example uses variables. You can also use a constant for the input variable.

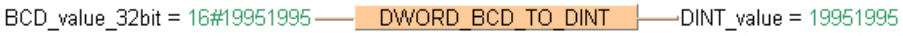
BCD constants can be indicated in Control FPWIN Pro as follows:

2#00011001100101010001100110010101 or 16#19951995

### POU body

**BCD_value_32bit** of the data type DOUBLE WORD is converted into a DOUBLE INTEGER value. The converted value is written into **DINT_value**.

## LD body


BCD_value_32bit = 16#19951995
DWORD_BCD_TO_DINT
DINT_value = 19951995

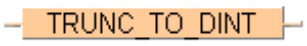
# TRUNC_TO_DINT

Truncate (cut off) decimal digits of REAL or LREAL input variable, convert to DOUBLE INTEGER

**TRUNC_TO_DINT** cuts off the decimal digits of a REAL, LREAL number and delivers an output variable of the data type DOUBLE INTEGER.

Note

- If the decimal digits are cut off, positive numbers will be decreased towards zero and negative numbers will be increased towards zero.
- Since REAL numbers only have a resolution of about 7 digits, information for large numbers will be lost.




## Parameters

Input

**Unnamed input** (REAL, LREAL)

Input data type

Output

**Unnamed output** (DINT)

Conversion result

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

if input variable does not have the data type REAL

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

output variable is greater than a 32-bit DOUBLE INTEGER

**sys_blsCarry** (turns to TRUE for one scan)

if output variable is zero

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

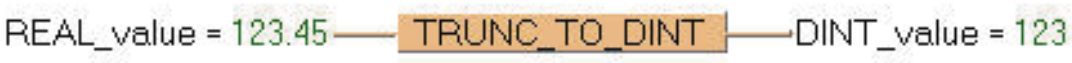
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | REAL_value | REAL | 0.0 | number betw. -2147483.000 ... +2147483.000 |
| 1 | VAR | DINT_value | DINT | 0 | number betw. -2147483 ... +2147483 |

This example uses variables. You can also use a constant for the input variable.

### POU body

The decimal digits of **REAL_value** are cut off. The result is stored as a 32-bit DOUBLE INTEGER in **DINT_value**.

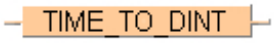
### LD body



# TIME_TO_DINT

TIME into DOUBLE INTEGER

**TIME_TO_DINT** converts a value of the data type TIME into a value of the data type DINT. The time 10ms corresponds to the value 1, e.g. an input value of T#1m0s is converted to the value 6000.




## Parameters

Input

**Unnamed input** (TIME)

Input data type

Output

**Unnamed output** (DINT)

Conversion result

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | time_value | TIME | T100ms |
| 1 | VAR | DINT_value | DINT | 0 |

This example uses variables. You can also use a constant for the input variable.

### POU body

**time_value** of the data type TIME is converted to value of the data type DOUBLE INTEGER. The result is written into the output variable **DINT_value**.

### LD body

# STRING_TO_DINT

STRING (decimal format) into DOUBLE INTEGER

This function converts a string in decimal formal into a value of the data type DINT.

At first the string is converted into a value of the data type STRING[32]. Finally this is converted into a value of the data type DINT in a subprogram of approximately 270 steps, which is also used by the functions STRING_TO_INT, STRING_TO_WORD, **STRING_TO_DINT** and **STRING_TO_DWORD**.


## Parameters

Input

**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (DINT)

Conversion result

## Remarks

If the required data type is supported, we recommend using the overloaded instruction TO_DINT

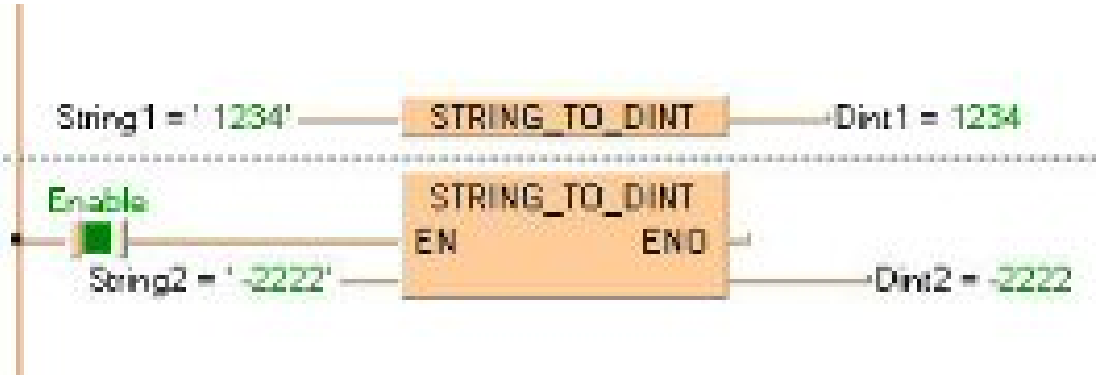
Acceptable format: '[Space][Sign][Decimal number][Space]' e.g. ' 123456 '

Acceptable characters:

- Space
  Space "␣"
- Sign
  "+“ (plus), "-" (minus)
- Decimal numbers
  Decimal numbers "0 - 9“

The analysis ends with the first non-decimal number.

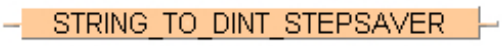
**Example with and without EN/ENO:**



# STRING_TO_DINT_STEPSAVER

STRING (decimal format right-justified) into DOUBLE INTEGER

This function converts a right-justifed decimal number in a string into a value of the data type DINT.




## Parameters

Input

**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (DINT)

Conversion result

## Remarks

The basic instruction **F78_DA2BIN** with approx. 11 steps is used. The PLC delivers an operation error especially when a character appears that is not a decimal number “0 - 9”, not a “+” or “-“ or not a space.

### Example


Acceptable Format:

'[Space][Sign][Decimal number]' e.g. ' 123456'

Acceptable characters:

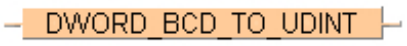
| | |
|---|---|
| Space | Space “␣“ |
| Signs | Plus "+“ and minus "-" |
| Decimal Numbers | Decimal numbers "0“ - "9“ |

## 12.15 Conversion to UDINT

# DWORD_BCD_TO_UDINT

Binary value of DOUBLE WORD into unsigned DOUBLE INTEGER

**DWORD_BCD_TO_UDINT** converts a binary value of the data type DWORD into a value of the data type Unsigned DOUBLE INTEGER.




## Parameters

Input

**Unnamed input** (DWORD_BCD)

input data type

Output

**Unnamed output** (UDINT)

Conversion result

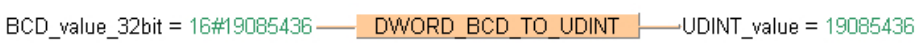
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UDINT_value | UDINT | 419976246 |
| 1 | VAR | BCD_value_32bit | DWORD | 16#19085436 |

### LD body



# TRUNC_TO_UDINT

Truncate (cut off) decimal digits of REAL or LREAL input variable, convert to unsigned DOUBLE INTEGER

**TRUNC_TO_UDINT** cuts off the digits following the decimal of a REAL, LREAL number and delivers an output variable of the data type Unsigned DOUBLE INTEGER.

Note

- If the decimal digits are cut off, positive numbers will be decreased towards zero and negative numbers will be increased towards zero.
- Since REAL numbers only have a resolution of about 7 digits, information for large numbers will be lost.


## Parameters

Input

**Unnamed input** (REAL, LREAL)

Input data type

Output

**Unnamed output** (UDINT)

Conversion result

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

if input variable does not have the data type REAL

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

output variable is greater than a 32-bit DOUBLE INTEGER

**sys_blsCarry** (turns to TRUE for one scan)

if output variable is zero

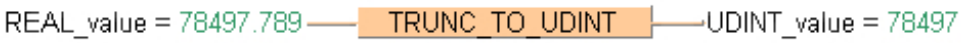
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UDINT_value | UDINT | 0 |
| 1 | VAR | REAL_value | REAL | 78497.79 |

### LD body



# STRING_TO_UDINT

STRING (decimal format) into unsigned DOUBLE INTEGER

**STRING_TO_UDINT** converts a string in decimal format into a value of the data type Unsigned DOUBLE INTEGER.

First, the string is converted into a value of the data type STRING[32], which is subsequently converted into a value of the data type UDINT in a subprogram with approximately 270 steps. This subprogram is also used by the functions **STRING_TO_INT**, **STRING_TO_WORD**, **STRING_TO_UDINT** and **STRING_TO_DWORD**.


## Remarks

Acceptable format: '[Space][Sign][Decimal number][Space]' e.g. ' 123456 '

Acceptable characters:

- Space
  Space "␣"
- Sign
  "+“ (plus), "-" (minus)
- Decimal numbers
  Decimal numbers "0 - 9“

The analysis ends with the first non-decimal number.

## Parameters

Input

**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (UDINT)

Conversion result

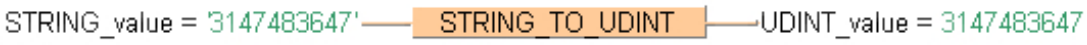
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UDINT_value | UDINT | 0 |
| 1 | VAR | STRING_value | STRING[... | '3147483647' |

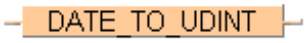
### LD body



# DATE_TO_UDINT

DATE into unsigned DOUBLE INTEGER

**DATE_TO_UDINT** converts a value of the data type DATE into a value of the data type Unsigned DOUBLE INTEGER according to its internal format, which is seconds elapsed since "2001-01-01".




## Parameters

Input

**Unnamed input** (DATE)

input data type

Output

**Unnamed output** (UDINT)

Conversion result

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

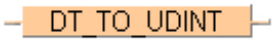
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UDINT_value | UDINT | 0 |
| 1 | VAR | DATE_value | DATE | D#2008-05-12 |

### LD body

# DT_TO_UDINT

DATE_AND_TIME into unsigned DOUBLE INTEGER

**DT_TO_UDINT** converts a value of the data type DATE_AND_TIME into a value of the data type Unsigned DOUBLE INTEGER according to its internal format, which is seconds elapsed since "2001-01-01-00:00:00".




## Parameters

Input

**Unnamed input** (DT)

input data type

Output

**Unnamed output** (UDINT)

Conversion result

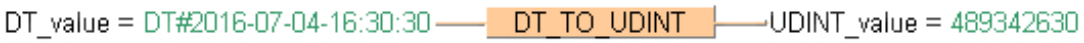
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UDINT_value | UDINT | 0 |
| 1 | VAR | DT_value | DATE_AND_TIME | DT#2016-07-04-16:30:30 |

### LD body



# TOD_TO_UDINT

TIME_OF_DAY into unsigned DOUBLE INTEGER

**TOD_TO_UDINT** converts a value of the data type TIME_OF_DAY into a value of the data type Unsigned DOUBLE INTEGER according to its internal format, which is seconds elapsed since "00:00:00".


## Parameters

Input

**Unnamed input** (TOD)

input data type

Output

**Unnamed output** (UDINT)

Conversion result

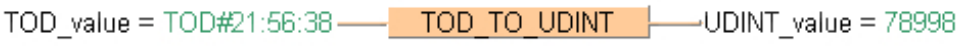
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UDINT_value | UDI... | 0 |
| 1 | VAR | TOD_value | TOD | TOD#21:56:38 |

## LD body



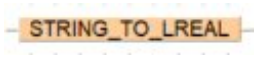
## 12.16 Conversion to LREAL

# STRING_TO_LREAL

STRING into LREAL

**STRING_TO_LREAL** converts a string in floating-point format into a value of the data type LREAL.

Thereby the attached string is first converted into a value of the data type STRING[64]. Finally this is converted into a value of the data type LREAL via a sub-program that requires approximately 580 steps.




**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (LREAL)

Conversion result

## Remarks

Permissible format: '[Space][Sign][Decimal numbers].[Decimal numbers][Space]' e.g. ' 1.45620752 '

Permissible characters:

| | |
|---|---|
| Space | Space "␣" |
| Decimal numbers | Decimal numbers "0"-"9" |

The analysis ends with the first non-decimal number.

## Example

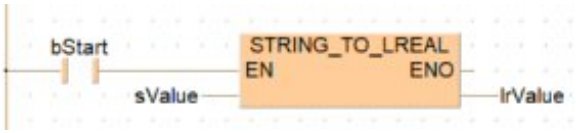
## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | lrValue | LREAL | 0.0 |
| 2 | VAR | bStart | BOOL | FALSE |
| 3 | VAR | sValue | STRING[64] | "" |

### LD body

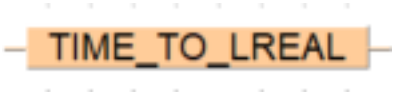
When the variable **bStart** is set to TRUE, the function is carried out.



# TIME_TO_LREAL

TIME into LREAL

**TIME_TO_LREAL** converts a value of the data type TIME into a value of the data type LREAL. 10ms of the data type TIME correspond to 1.0 REAL unit, e.g. when TIME = 10ms, REAL = 1.0; when TIME = 1s, REAL = 100.0. The resolution amounts to 10ms.




**Unnamed input** (TIME)

Input data type

Output

**Unnamed output** (LREAL)

Conversion result

## Example

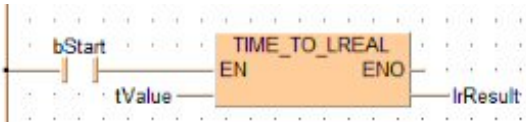
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bStart | BOOL | FALSE |
| 2 | VAR | tValue | TIME | T#0s |
| 3 | VAR | lrResult | REAL | 0 |

### LD body

When the variable **bStart** is set to TRUE, the function is carried out.

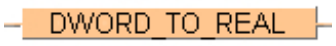


## 12.17 Conversion to REAL

# DWORD_TO_REAL

DOUBLE WORD into REAL

**DWORD_TO_REAL** moves the bit set information of a DWORD variable to a REAL variable. The same functionality can be obtained using **DWORD_OVERLAPPING_DUT**.




## Parameters

Input

**Unnamed input** (DWORD)

input data type

Output

**Unnamed output** (REAL)

Conversion result

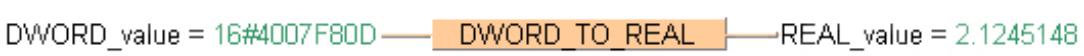
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | REAL_value | REAL | 0.0 |
| 1 | VAR | DWORD_value | DWORD | 16#4007F80D |

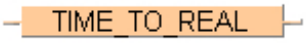
### LD body



# TIME_TO_REAL

TIME into REAL

**TIME_TO_REAL** converts a value of the data type TIME into a value of the data type REAL. 10ms of the data type TIME correspond to 1.0 REAL unit, e.g. when TIME = 10ms, REAL = 1.0; when TIME = 1s, REAL = 100.0. The resolution amounts to 10ms.




## Parameters

Input

**Unnamed input** (TIME)

input data type

Output

**Unnamed output** (REAL)

Conversion result

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_time | TIME | T#1h1m1s | |
| 1 | VAR | result_time | REAL | 0.0 | result: here 366100.0 |

This example uses variables. You can also use a constant for the input variable.

## LD body

# STRING_TO_REAL

STRING into REAL

**STRING_TO_REAL** converts a string in floating-point format into a value of the data type REAL.

Thereby the attached string is first converted into a value of the data type STRING[64]. Finally this is converted into a value of the data type REAL via a sub-program that requires approximately 290 steps.


## Parameters

Input

**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (REAL)

Conversion result

## Remarks

Permissible format: '[Space][Sign][Decimal numbers].[Decimal numbers][Space]' e.g. ' -123.456 '

Permissible characters:

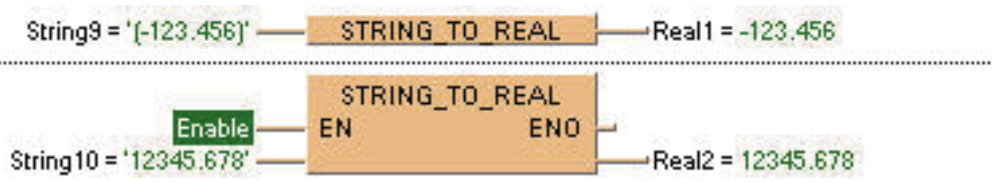
- Space

  Space "␣"
- Decimal numbers

  Decimal numbers "0"-"9"

The analysis ends with the first non-decimal number.

### Example with and without EN/ENO:



## 12.18 Conversion to TIME

# WORD_TO_TIME

WORD in TIME

**WORD_TO_TIME** converts a value of the data type WORD into a value of the data type TIME.


## Parameters

Input

**Unnamed input** (WORD)

Input data type

Output

**Unnamed output** (TIME)

Conversion result

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

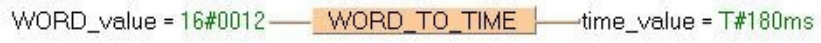
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | WORD_value | WORD | 0 |
| 1 | VAR | time_value | TIME | T#0s |

This example uses variables. You can also use a constant for the input variable.

### POU body

**WORD_value** of the data type WORD (16-bit) is converted into a value of the data type TIME (16-bit). The result will be written into the output variable **time_value**.

### LD body



# DWORD_TO_TIME

DOUBLE WORD in TIME

**DWORD_TO_TIME** converts a value of the data type DWORD into a value of the data type TIME. A value of 1 corresponds to a time of 10ms, e.g. the input value 12345 (16#3039) is converted to a TIME T#2m3s450.00ms.


## Parameters

Input

**Unnamed input** (DWORD)

Input data type

Output

**Unnamed output** (TIME)

Conversion result

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

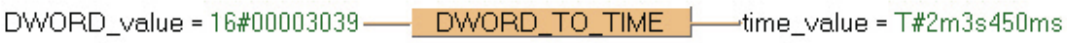
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | DWORD_value | DWORD | 0 | example value: 16#3039 |
| 1 | VAR | time_value | TIME | T#0s | result: T#2m3s450.00ms |

This example uses variables. You can also use a constant for the input variable.

## POU body

**DWORD_value** of the data type DWORD (32-bit) is converted to value of the data type TIME (16-bit). The result is written into the output variable **time_value**.

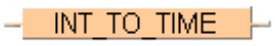
## LD body



# INT_TO_TIME

INTEGER into TIME

**INT_TO_TIME** converts a value of the data type INT into a value of the data type TIME. The resolution is 10ms, e.g. when the INT value = 350, the TIME value = 3s500ms.




## Parameters

Input

**Unnamed input** (INT)

Input data type

Output

**Unnamed output** (TIME)

Conversion result

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | tme_value | TIME | T#0s |
| 1 | VAR | INT_value | INT | 0 |

This example uses variables. You can also use a constant for the input variable.

### POU body

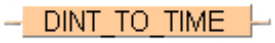
**INT_value** of the data type INTEGER is converted into a value of the data type TIME. The result will be written into the output variable **time_value**.

### LD body

# DINT_TO_TIME

DOUBLE INTEGER into TIME

**DINT_TO_TIME** converts a value of the data type DINT into a value of the data type TIME. A value of 1 corresponds to a time of 10ms, e.g. an input value of 123 is converted to a TIME T#1s230.00ms.




## Parameters

Input

**Unnamed input** (DINT)

Input data type

Output

**Unnamed output** (TIME)

Conversion result

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | DINT_value | DINT | 0 | |
| 1 | VAR | TIME_value | TIME | T#0s | result: T#1s230.00ms |

This example uses variables. You can also use a constant for the input variable.

## POU body

**DINT_value** of the data type DOUBLE INTEGER is converted to value of the data type TIME. The result is written into the output variable **time_value**.

## LD body

# LREAL_TO_TIME

LREAL into TIME

**LREAL_TO_TIME** converts a value of the data type LREAL into a value of the data type TIME. 10ms of the data type TIME correspond to 1.0 LREAL unit, e.g. when LREAL = 1.0, TIME = 10ms; when LREAL = 100.0, TIME = 1s. The value of the data type real is rounded off to the nearest whole number for the conversion.


**Unnamed input** (LREAL)

Input data type

Output

**Unnamed output** (TIME)

Conversion result

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bStart | BOOL | FALSE |
| 2 | VAR | lrValue | LREAL | 0 |
| 3 | VAR | tResult | TIME | T#0s |

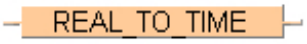
### LD body

When the variable **bStart** is set to TRUE, the function is carried out.

# REAL_TO_TIME

REAL into TIME

**REAL_TO_TIME** converts a value of the data type REAL into a value of the data type TIME. 10ms of the data type TIME correspond to 1.0 REAL unit, e.g. when REAL = 1.0, TIME = 10ms; when REAL = 100.0, TIME = 1s. The value of the data type real is rounded off to the nearest whole number for the conversion.




## Parameters

Input

**Unnamed input** (REAL)

Input data type

Output

**Unnamed output** (TIME)

Conversion result

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | result_time | TIME | T#0s |

## POU body

By clicking on the monitor icon while in the online mode, you can see the result 0.00ms immediately. Since the value at the REAL input contact is less than 0.5, it is rounded down to 0.0.

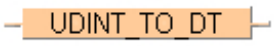
## LD body

## 12.19 Conversion to DATE_AND_TIME (DT)

# UDINT_TO_DT

Unsigned DOUBLE INTEGER into DATE_AND_TIME

**UDINT_TO_DT** converts a value of the data type Unsigned DOUBLE INTEGER into a value of the data type DATE_AND_TIME.




## Parameters

Input

**Unnamed input** (UDINT)

Input data type

Output

**Unnamed output** (DATE_AND_TIME)

Conversion result

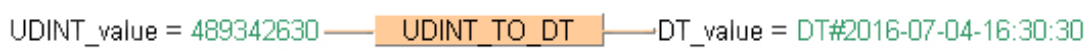
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UDINT_value | UDINT | 489342630 |
| 1 | VAR | DT_value | DATE_AND_TIME | DT#2001-01-01-00:00:00 |

### LD body

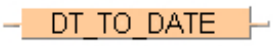


## 12.20 Conversion to DATE

# DT_TO_DATE

DATE_AND_TIME into DATE

**DT_TO_DATE** converts a value of the data type DATE_AND_TIME into a value of the data type DATE.




## Parameters

Input

**Unnamed input** (DATE_AND_TIME)

Input data type

Output

**Unnamed output** (DATE)

Conversion result

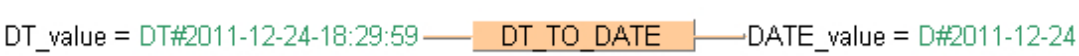
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DT_value | DATE_AND_TIME | DT#2011-12-24-18:29:59 |
| 1 | VAR | DATE_value | DATE | D#2001-01-01 |

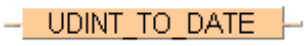
### LD body



# UDINT_TO_DATE

Unsigned DOUBLE INTEGER into DATE

**UDINT_TO_DATE** converts a value of the data type Unsigned DOUBLE INTEGER into a value of the data type DATE.




## Parameters

Input

**Unnamed input** (UDINT)

Input data type

Output

**Unnamed output** (DATE)

Conversion result

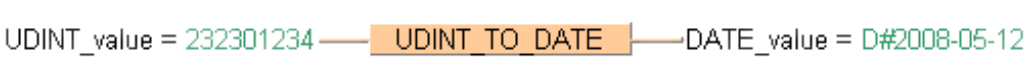
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UDINT_value | UDINT | 232301234 |
| 1 | VAR | DATE_value | DATE | D#2001-01-01 |

### LD body

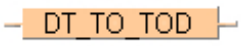


## 12.21 Conversion to TIME_OF_DAY (TOD)

# DT_TO_TOD

DATE_AND_TIME into TIME_OF_DAY

**DT_TO_TOD** converts a value of the data type DATE_AND_TIME into a value of the data type TIME_OF_DAY.




## Parameters

Input

**Unnamed input** (DATE_AND_TIME)

Input data type

Output

**Unnamed output** (TIME_OF_DAY)

Conversion result

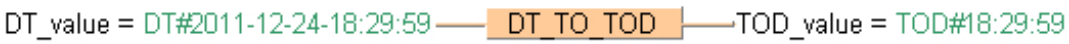
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DT_value | DATE_AND_TIME | DT#2011-12-24-18:29:59 |
| 1 | VAR | TOD_value | TOD | TOD#00:00:00 |

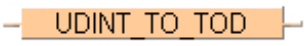
### LD body



# UDINT_TO_TOD

Unsigned DOUBLE INTEGER into TIME_OF_DAY

**UDINT_TO_TOD** converts a value of the data type Unsigned DOUBLE INTEGER into a value of the data type TIME_OF_DAY.




## Parameters

Input

**Unnamed input** (UDINT)

Input data type

Output

**Unnamed output** (TIME_OF_DAY)

Conversion result

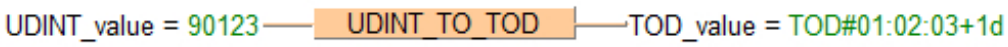
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UDINT_value | UDINT | 165398 |
| 1 | VAR | TOD_value | TOD | TOD#00:00:00 |

### LD body

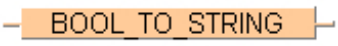


## 12.22 Conversion to STRING

# BOOL_TO_STRING

BOOL into STRING

The function **BOOL_TO_STRING** converts a value of the data type BOOL into a value of the data type STRING[2]. The resulting string is represented by ' 0' or ' 1'.




## Parameters

Input

**Unnamed input** (BOOL)

Input data type

Output

**Unnamed output** (STRING)

Conversion result

## Remarks

When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

## Example result string = '0' or '1'

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | BOOL | TRUE | example value |
| 1 | VAR | result_string | STRING[2] | '' | result: here '1' |

The input variable **input_value** of the data type BOOL is intialized by the value TRUE. The output variable **result_string** is of the data type STRING[2]. It can store a maximum of two characters. You can declare a character string that has more than one character, e.g. STRING[5]. From the 5 characters reserved, only 2 are used.

Instead of using the variable **input_value**, you can write the constants TRUE or FALSE directly to the function’s input contact in the body.

### POU body

The **input_value** of the data type BOOL is converted into STRING[2]. The converted value is written to **result_string**. When the variable **input_value** = TRUE, **result_string** shows ' 1'.

### LD body


## Example result string = 'TRUE' or 'FALSE'

### POU header

If you wish to have the result 'TRUE' or 'FALSE' instead of ' 0' or ' 1', you cannot use the function **BOOL_TO_STRING**. This example illustrates how you create a STRING[5] that contains the characters 'TRUE' or 'FALSE' from an input value of the data type BOOL.All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

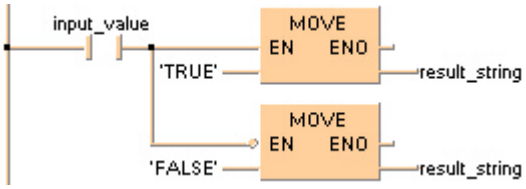
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | BOOL | TRUE | example value |
| 1 | VAR | result_string | STRING[5] | '' | result: here 'TRUE' |

In this example, both an input variable **input_value** of the data type BOOL and an output variable **result_string** of the data type STRING[5] are declared.

### POU body

In order to realize the intended operation, the standard function **MOVE** is used. It assigns the value of its input to its output unchanged. At the input, the STRING constant 'TRUE' or 'FALSE' is attached. In essence a "BOOL to STRING" conversion occurs, since the Boolian variable **input_variable** at the enable input (EN) contact decides the output of STRING.

### LD body

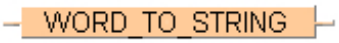


# WORD_TO_STRING

WORD into STRING

The function **WORD_TO_STRING** converts a value of the data type WORD into a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.




## Parameters

Input

**Unnamed input** (WORD)

Input data type

Output

**Unnamed output** (STRING)

Conversion result

## Remarks

When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

### Example

| Input | Output defined as | Results in |
|---|---|---|
| 16#ABCD | STRING[1] | 'D' |
| | STRING[2] | 'CD' |
| | STRING[3] | 'BCD' |
| | STRING[4] | 'ABCD' |
| | STRING[5] | '0ABCD' |
| | STRING[6] | '00ABCD' |
| | and so on... | |

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

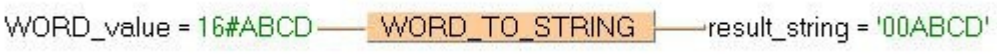
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | WORD | 0 | example value |
| 1 | VAR | result_string | STRING[6] | '' | result: here '00ABCD' |

The input variable **input_value** of the data type WORD is intialized by the value 16#ABCD. The output variable **result_string** is of the data type STRING[6]. It can store a maximum of 6 characters. Instead of using the variable **input_value**, you can enter a constant directly at the function's input contact in the body.

### POU body

The **input_value** of the data type WORD is converted into STRING[6]. The converted value is written to **result_string**. When the variable **input_value** = 16#ABCD, **result_string** shows '00ABCD'.

### LD body




## Example

### POU header

This example illustrates how you create STRING[4] out of the data type WORD in which the leading part of the string '16#' is cut out.

All input and output variables used for programming this function have been declared in the POU header.  The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | WORD | 16#1234 | example value |
| 1 | VAR | result_string1 | STRING[7] | '' | result: here '0001234' |
| 2 | VAR | result_string | STRING[4] | '' | result: here '1234' |

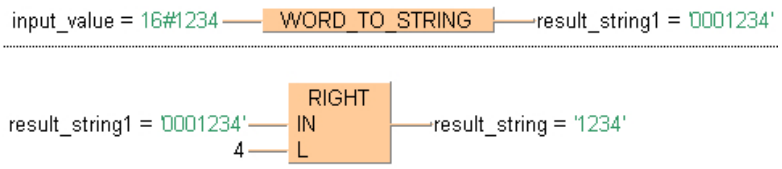
In this example, both an input variable **input_value** of the data type WORD and an output variable **result_string** of the data type STRING[4] are declared.

### POU body

In carrying out the operation in question, the standard function **RIGHT** is attached to the function **WORD_TO_STRING**. **RIGHT** creates a right-justified character string of length **L**.

In the example, the output string of **WORD_TO_STRING** function is added at the input of the **RIGHT** function. At the **L** input of **RIGHT**, the INT constant 4 determines the length of the STRING to be replaced. Out of the variable **input_value** = 0001234, the result_string 1234 results from the data type conversion and the **RIGHT** function.

### LD body

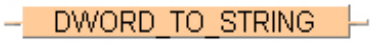


# DWORD_TO_STRING

DOUBLE WORD into STRING

The function **DWORD_TO_STRING** converts a value of the data type DWORD into a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.




## Parameters

Input

**Unnamed input** (DWORD)

Input data type

Output

**Unnamed output** (STRING)

Conversion result

## Remarks

When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

### Example

| Input | Output defined as | Results in |
|---|---|---|
| 16#ABCDEFFE | STRING[2] | 'FE' |
| | STRING[4] | 'EFFE' |
| | STRING[6] | 'CDEFFE' |
| | STRING[8] | 'ABCDEFFE' |
| | STRING[10] | '00ABCDEFFE' |
| | STRING[12] | '0000ABCDEFFE' |
| | and so on... | |

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

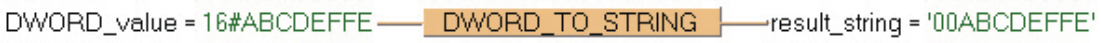
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | DWORD_value | DWORD | 0 | example value: 16#ABCDEFFE |
| 1 | VAR | result_string | STRING[10] | '' | result: '00ABCDEFFE' |

The input variable **DWORD_value** of the data type DWORD is intialized by the value 16#ABCDEFFE. The output variable **result_string** is of the data type STRING[10]. It can store a maximum of 10 characters. Instead of using the variable **DWORD_value**, you can enter a constant directly at the function's input contact in the body.

### POU body

The **DWORD_value** of the data type DWORD is converted into STRING[10]. The converted value is written to **result_string**. When the variable **DWORD_value** = 16#ABCDEFFE, **result_string** shows '00ABCDEFFE'.

### LD body




## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

This example illustrates how you create STRING[10] out of the data type DWORD in which the leading part of the string '16#' is replaced by the string '0x'.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | DWORD | 16#12345678 | example value |
| 1 | VAR | result_string | STRING[10] | '' | result: here '0x12345678' |

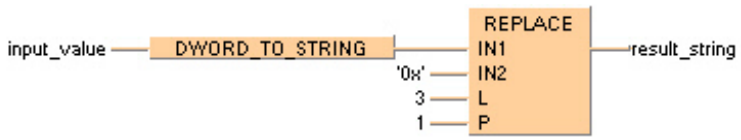
In this example the input variable **input_value** of the data type DWORD and an output variable **result_string** of the data type STRING[10] are declared.

### POU body

In carrying out the operation in question, the standard function **REPLACE** is attached to the function **DWORD_TO_STRING**. **REPLACE** replaces one section of a character string with another.

In the example, the output string of **DWORD_TO_STRING** function is added at input **IN1** of the **REPLACE** function. At input **IN2**, the STRING constant '0x' is added as the replacement STRING. At the **L** input of **REPLACE**, the INT constant 3 determines the length of the STRING to be replaced. The **P** input determines the position at which the replacement begins. In this case it is the INT number 1. From the variable **input_value** = 16#12345678, the **result_string** = '0x12345678' results after undergoing the data type conversion and **REPLACE** function.

### LD body

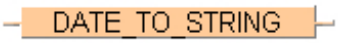


# DATE_TO_STRING

DATE into STRING

**DATE_TO_STRING** converts a value of the data type DATE into a value of the data type STRING[10].

The range for the input date is from D#2001-01-01 to D#2099-12-31.




## Parameters

Input

**Unnamed input** (DATE)

Input data type

Output

**Unnamed output** (STRING)

Conversion resultSTRING[10]

## Remarks

All character spaces in the result string will be filled.

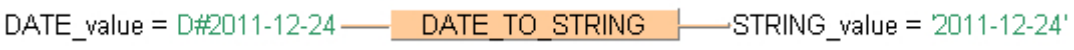
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DATE_value | DATE | D#2011-12-24 |
| 1 | VAR | STRING_value | STRING[10] | '' |

### LD body

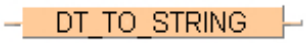


# DT_TO_STRING

DATE_AND_TIME into STRING

**DT_TO_STRING** converts a value of the data type DATE_AND_TIME into a value of the data type STRING[19].

The range for the input date is from DT#2001-01-01-00:00:00 to DT#2099-12-31-23:59:59.




## Parameters

Input

**Unnamed input** (DATE_AND_TIME)

Input data type

Output

**Unnamed output** (STRING)

Conversion resultSTRING[19]

## Remarks

All character spaces in the result string will be filled.

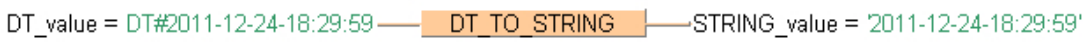
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DT_value | DATE_AND_TIME | DT#2011-12-24-18:29:59 |
| 1 | VAR | STRING_value | STRING[19] | '' |

### LD body

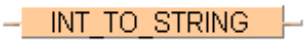


# INT_TO_STRING

INTEGER into STRING

The function **INT_TO_STRING** converts a value of the data type INT into a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.




## Parameters

Input

**Unnamed input** (INT)

Input data type

Output

**Unnamed output** (STRING)

Conversion result

## Remarks

When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

### Example

| Function used | String1 defined as | Result |
|---|---|---|
| **String1**:=INT_TO_STRING(-12345) | STRING[1] | '5' |
| | STRING[2] | '45' |
| | STRING[3] | '345' |
| | STRING[4] | '2345' |
| | STRING[5] | '12345' |
| | STRING[6] | '-12345' |
| | STRING[7] | ␣ |
| | STRING[8] | ␣␣ |
| | and so on... | |

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | INT_value | INT | -12345 | example value |
| 1 | VAR | result_string | STRING[8] | '' | result: here '-12345' |

The input variable **INT_value** of the data type INT is intialized by the value -12345. The output variable **result_string** is of the data type STRING[8]. It can store a maximum of 8 characters. Instead of using the variable **INT_value**, you can enter a constant directly at the function's input contact in the body.

### POU body

The **INT_value** of the data type INT is converted into STRING[8]. The converted value is written to **result_string**. When the variable **INT_value** = -12345, **result_string** shows '␣␣-12345'.

### LD body


## Example

### POU header

This example illustrates how you create a STRING[2] that appears right justified out of the data type INT.

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | INT_value | INT | 12 | example value |
| 1 | VAR | result_string | STRING[2] | '' | result: here '12' |

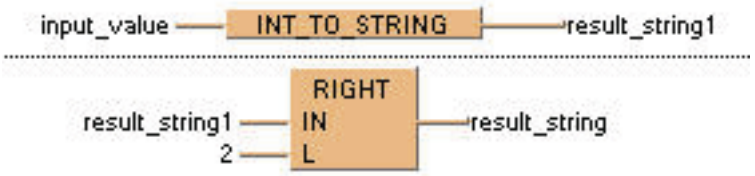
In this example, both an input variable **INT_value** of the data type INT and an output variable **result_string** of the data type STRING[2] are declared.

## POU body

In carrying out the operation in question, the standard function **RIGHT** is attached to the function **INT_TO_STRING**. **RIGHT** creates a right-justified character string with the length **L**.

In the example, the variable **INT_variable** = 12 is converted by **INT_TO_STRING** to the dummy string '␣␣12'. The function **RIGHT** then creates the **result_string** '12'.

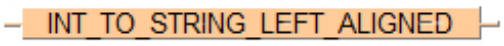
## LD body



# INT_TO_STRING_LEFT_ALIGNED

INTEGER into left aligned STRING

It converts a value of the data type INT into a value of the data type STRING in left-aligned decimal representation.




## Parameters

Input

**Unnamed input** (INT)

input data type

Output

**Unnamed output** (STRING)

conversion result

## Remarks

When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

### Example

| Function used | String1 defined as | Result |
|---|---|---|
| `String1:= INT_TO_STRING_LEFT_ALIGNED(-12345)` | STRING[1] | '-' |
| | STRING[2] | '-1' |
| | STRING[3] | '-12' |
| | STRING[4] | '-123' |
| | STRING[5] | '-1234' |
| | STRING[6] | '-12345' |
| | STRING[7] | '-12345' |
| | STRING[8] | '-12345' |
| | and so on... | |

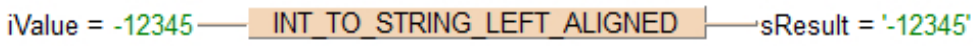
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | iValue | INT | -12345 | example value |
| 1 | VAR | sResult | STRING[8] | '' | result: here '-12345' |

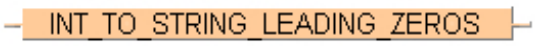
### LD body



# INT_TO_STRING_LEADING_ZEROS

INTEGER into STRING

The function **INT_TO_STRING_LEADING_ZEROS** converts a value of the data type INT (positive values) into a value of the data type STRING.Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.




## Parameters

Input

**Unnamed input** (INT)

input data type (only positive values)

Output

**Unnamed output** (STRING)

conversion result

## Remarks

- Use only positive integer values. For negative values, please use INT_TO_STRING.
- When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

### Example


| Function used | String1 defined as | Result |
|---|---|---|
| String1:=INT_TO_STRING(25) | STRING[1] | '5' |
| | STRING[2] | '25' |
| | STRING[3] | '025' |

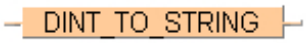
| Function used | String1 defined as | Result |
|---|---|---|
| | STRING[4] | '0025' |
| | STRING[5] | '00025' |
| | STRING[6] | '000025' |
| | STRING[7] | '0000025' |
| | STRING[8] | '00000025' |
| | and so on... | |

# DINT_TO_STRING

DOUBLE INTEGER into STRING

The function **DINT_TO_STRING** converts a value of the data type DINT into a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.




## Parameters

Input

**Unnamed input** (DINT)

Input data type

Output

**Unnamed output** (STRING)

Conversion result

## Remarks

When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

### Example

| Function used | String1 defined as | Result |
|---|---|---|
| String1:= DINT_TO_STRING(-12345678) | STRING[2] | '78' |
| | STRING[4] | '5678' |
| | STRING[6] | '345678' |
| | STRING[8] | '12345678' |
| | STRING[10] | ␣ |
| | STRING[12] | '<br>␣␣<br>␣<br>-12345678' |
| | and so on... | |

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_string | DINT | 12345678 | example value |
| 1 | VAR | result_string | STRING[11] | '' | result: here '12345678' |

The input variable **input_value** of the data type DINT is intialized by the value 12345678. The output variable **result_string** is of the data type STRING[11]. It can store a maximum of 11 characters. Instead of using the variable **input_value**, you can enter a constant directly at the function's input contact in the body.

### POU body

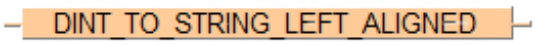
The **input_value** of the data type DINT is converted into STRING[11]. The converted value is written to **result_string**. When the variable **input_value** = 12345678, **result_string** shows '␣␣␣12345678'.

### LD body

# DINT_TO_STRING_LEFT_ALIGNED

DOUBLE INTEGER into left-aligend STRING

It converts a value of the data type DINT into a value of the data type STRING in left-aligned decimal representation.




## Parameters

Input

**DINT**

Input data type

Output

**STRING**

Conversion result

Explanation

| Function used | String1 defined as | Result |
|---|---|---|
| String1:= DINT_TO_STRING_LEFT_ALIGNED(-12345678) | STRING[2] | '-1' |
| | STRING[4] | '-123' |
| | STRING[6] | '-12345' |
| | STRING[8] | '-1234567' |
| | STRING[10] | '-12345678' |
| | STRING[12] | '-12345678' |
| | and so on... | |

Note

When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

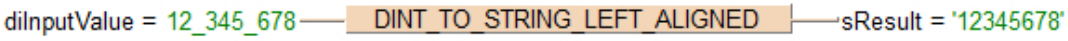
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | diInputValue | DINT | 12345678 |
| 1 | VAR | sResult | STRING[11] | '' |

### LD body



# DINT_TO_STRING_LEADING_ZEROS

DOUBLE INTEGER into STRING

This function converts a value of the data type DINT (positive value) into a value of the data type STRING. It generates a result string in decimal representation that is right aligned. It is filled with leading zeros up to the maximum number of characters defined for the string.


## Parameters

Input

**Unnamed input** (DINT)

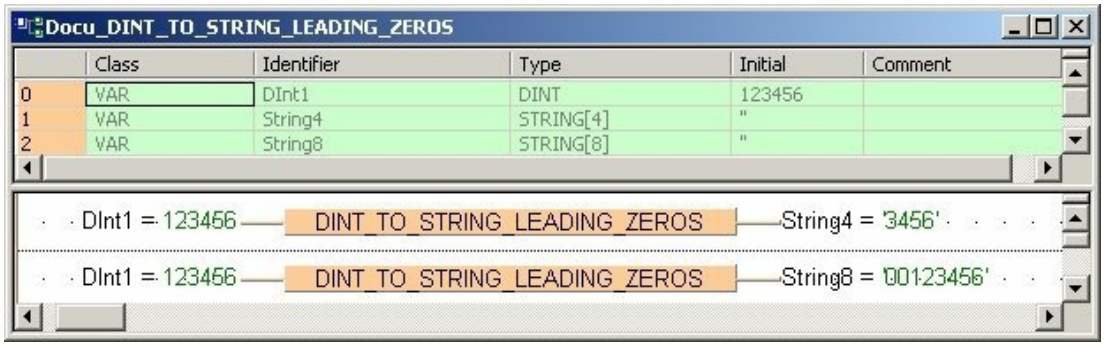
Input data type

Output

**Unnamed output** (STRING)

Conversion result

### Example




Explanation

| Function used | String1 defined as | Result |
|---|---|---|
| **String1**:=DINT_TO_STRING(12345678) | STRING[2] | '78' |
| | STRING[4] | '5678' |
| | STRING[6] | '345678' |
| | STRING[8] | '12345678' |
| | STRING[10] | '0012345678' |
| | STRING[12] | '000012345678' |
| | and so on... | |

Note

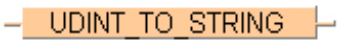
When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

# UDINT_TO_STRING

Unsigned DOUBLE INTEGER into STRING

The function **UDINT_TO_STRING** converts a value of the data type UDINT into a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.




## Parameters

Input

**Unnamed input** (UDINT)

Input data type

Output

**Unnamed output** (STRING)

Conversion result

## Remarks

When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

### Example

| Function used | String1 defined as | Result |
|---|---|---|
| String1:= UDINT_TO_STRING(-12345678) | STRING[2] | '78' |
| | STRING[4] | '5678' |
| | STRING[6] | '345678' |
| | STRING[8] | '12345678' |
| | STRING[10] | ␣ |
| | STRING[12] | '<br>␣␣<br>␣<br>-12345678' |
| | and so on... | |

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

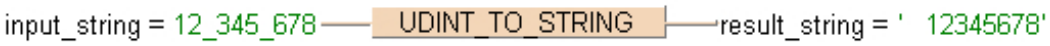
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_string | UDINT | 12345678 | example value |
| 1 | VAR | result_string | STRING[11] | '' | result: here '12345678' |

The input variable **input_value** of the data type UDINT is intialized by the value 12345678. The output variable **result_string** is of the data type STRING[11]. It can store a maximum of 11 characters. Instead of using the variable **input_value**, you can enter a constant directly at the function's input contact in the body.

### POU body

The **input_value** of the data type DINT is converted into STRING[11]. The converted value is written to **result_string**. When the variable **input_value** = 12345678, **result_string** shows '␣␣␣12345678'.

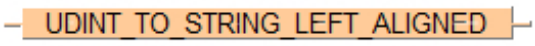
### LD body



# UDINT_TO_STRING_LEFT_ALIGNED

Unsigned DOUBLE INTEGER into left-aligned STRING

It converts a value of the data type UDINT into a value of the data type STRING in left-aligned decimal representation.




## Parameters

Input

**UDINT**

Input data type

Output

**STRING**

Conversion result

Remarks
Explanation

| Function used | String1 defined as | Result |
|---|---|---|
| String1:= UDINT_TO_STRING_LEFT_ALIGNED(-12345678) | STRING[2] | '-1' |
| | STRING[4] | '-123' |
| | STRING[6] | '-12345' |
| | STRING[8] | '-1234567' |
| | STRING[10] | '-12345678' |
| | STRING[12] | '-12345678' |
| | and so on... | |

### Note

When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

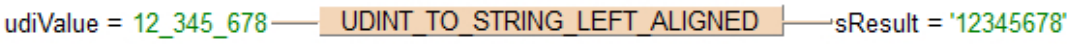
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | udiValue | UDINT | 12345678 | example value |
| 1 | VAR | sResult | STRING[11] | '' | result: here '12345678' |

### LD body



# UDINT_TO_STRING_LEADING_ZEROS

Unsigned DOUBLE INTEGER into STRING

This function converts a value of the data type UDINT (positive value) into a value of the data type STRING. It generates a result string in decimal representation that is right aligned. It is filled with leading zeros up to the maximum number of characters defined for the string.


## Parameters

Input

**Unnamed input** (UDINT)

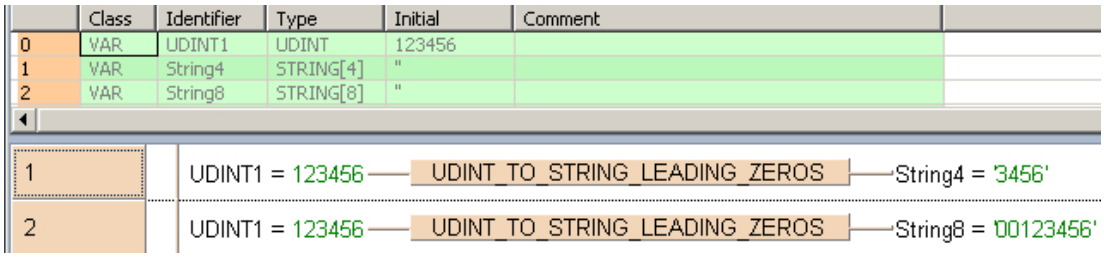
Input data type

Output

**Unnamed output** (STRING)

Conversion result

Example




Explanation:

| Function used | String1 defined as | Result |
|---|---|---|
| String1:=UDINT_TO_STRING(12345678) | STRING[2] | '78' |
| | STRING[4] | '5678' |
| | STRING[6] | '345678' |
| | STRING[8] | '12345678' |
| | STRING[10] | '0012345678' |
| | STRING[12] | '000012345678' |
| | and so on... | |

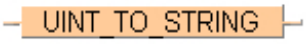
When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

# UINT_TO_STRING

Unsigned INTEGER into STRING

**UINT_TO_STRING** converts a value of the data type Unsigned INTEGER into a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.




## Parameters

Input

**Unnamed input** (UINT)

Input data type

Output

**Unnamed output** (STRING)

Conversion result

## Remarks

- The result is not specified when the range of the input values does not match the range of the output values.
- When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

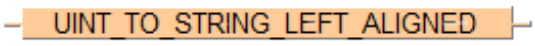
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UINT_value | UINT | 49152 |
| 1 | VAR | STRING_value | STRING[8] | '' |

**LD body**

# UINT_TO_STRING_LEFT_ALIGNED

Unsigned INTEGER into left-aligned STRING

It converts a value of the data type Unsigned INTEGER into a value of the data type STRING in left-aligned decimal representation.




## Parameters

Input

**Unnamed input** (UINT)

Input data type

Output

**Unnamed output** (STRING)

Conversion result

### Note

The result is not specified when the range of the input values does not match the range of the output values.

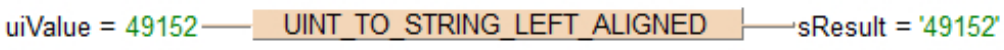
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | uiValue | UINT | 49152 |
| 1 | VAR | sResult | STRING[8] | '' |

## LD body



# UINT_TO_STRING_LEADING_ZEROS

Unsigned INTEGER into STRING

**UINT_TO_STRING_LEADING_ZEROS** converts a value of the data type Unsigned INTEGER into a value of the data type STRING.

Generates a result string in right-aligned decimal representation, filled with leading spaces up to the predefined maximum number of characters.


## Parameters

Input

**Unnamed input** (UINT)

Input data type

Output

**Unnamed output** (STRING)

Conversion result

## Remarks

- The result is not specified when the range of the input values does not match the range of the output values.
- When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

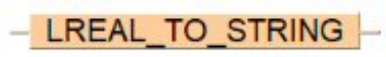
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UINT_value | UINT | 49152 |
| 1 | VAR | STRING_value | STRING[8] | '' |

### LD body

# LREAL_TO_STRING

LREAL into STRING

The function **LREAL_TO_STRING** converts a value from the data type LREAL into a value of the data type STRING[64], which has 7 spaces both before and after the decimal point. The resulting string is right justified within the range '-999999.0000000' to '9999999.0000000'. The plus sign is omitted in the positive range. Leading zeros are filled with empty spaces (e.g. out of -12.0 the STRING '␣␣-12.0').




**Unnamed input** (REAL, LREAL)

Input data type

Output

**Unnamed output** (STRING)

Conversion resultSTRING[15]

## Example

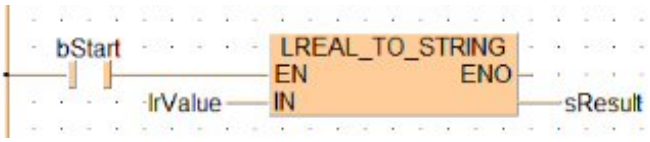
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | lrValue | LREAL | 5.983e5 |
| 2 | VAR | bStart | BOOL | FALSE |
| 3 | VAR | sResult | STRING[64] | '' |

### LD body

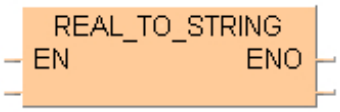
When the variable **bStart** is set to TRUE, the function is carried out.



# REAL_TO_STRING

REAL into STRING

The function **REAL_TO_STRING** converts a value from the data type REAL into a value of the data type STRING[15], which has 7 spaces both before and after the decimal point. The resulting string is right justified within the range '-999999.0000000' to '9999999.0000000'. The plus sign is omitted in the positive range. Leading zeros are filled with empty spaces (e.g. out of -12.0 the STRING '␣␣-12.0').




## Parameters

Input

**Unnamed input** (REAL)

Input data type

Output

**Unnamed output** (STRING)

Conversion resultSTRING[15]

## Remarks

- The function requires approximately 160 steps of program memory. For repeated use you should integrate it into a user function that is only stored once in the memory.
- When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_string | REAL | -123.4560166 | example value |
| 1 | VAR | result_string | STRING[15] | '' | result: here '-123.4560166' |

The input variable **input_value** of the data type REAL is intialized by the value -123.4560166. The output variable **result_string** is of the data type STRING[15]. It can store a maximum of 15 characters. Instead of using the variable **input_value**, you can enter a constant directly at the function's input contact in the body.

### POU body

The **input_value** of the data type REAL is converted into STRING[15]. The converted value is written to **result_string**. When the variable **input_value** = 123.4560166, **result_string** shows ' -123.4560165'.

### LD body


## Example

### POU header

This example illustrates how you create a STRING[7] with 4 positions before and 2 positions after the decimal point out of the data type REAL.

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_string | REAL | -123.4560166 | example value |
| 1 | VAR | result_string | STRING[7] | '' | result: here '-123.45' |

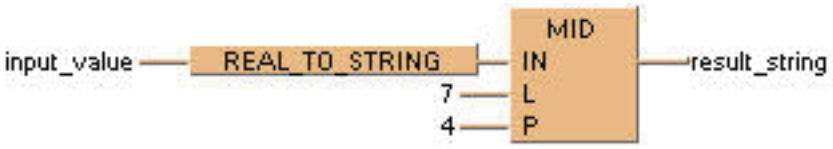
In this example, both an input variable **input_value** of the data type REAL and an output variable **result_string** of the data type STRING[7] are declared.

### POU body

In carrying out the operation in question, the standard function **MID** is attached to the function **REAL_TO_STRING**. **MID** creates a central sector in the character string from position **P** (INT value) with **L** (INT value) characters.

In the example, the INT constant 7 is entered at the **L** input of **MID**, which determines the length of the result string. The INT constant 4 at input **P** determines the position at which the central sector begins. Out of the variable **input_value** = -123.4560166, the STRING ' -123.4560166' results from the data type conversion. The **MID** function cuts off the STRING at position 4 and yields the **result_string** '-123.45'.

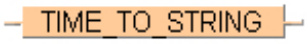
### LD body



# TIME_TO_STRING

TIME into STRING

The function **TIME_TO_STRING** converts a value of the data type TIME into a value of the data type STRING[20]. In accordance with IEC-1131, the result string is displayed with a short time prefix and without underlines. Possible values for the result string's range are from 'T#000d00h00m00s000ms' to 'T#248d13h13m56s470ms'.




## Parameters

Input

**Unnamed input** (TIME)

Input data type

Output

**Unnamed output** (STRING)

Conversion result

## Remarks

When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

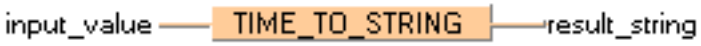
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | TIME | T#1h30m45s | example value |
| 1 | VAR | result_string | STRING[20] | '' | result: here 'T#000d01h30m45s000ms' |

The input variable **input_value** of the data type TIME is initialized by the value T#1h30m45s. The output variable **result_string** is of the data type STRING[20]. It can store a maximum of 20 characters. Instead of using the variable **input_value**, you can enter a constant directly at the function's input contact in the body.

### POU body

The **input_value** of the data type TIME is converted into STRING[20]. The converted value is written to **result_string**. When the variable **input_value** = T#1h30m45s, **result_string** shows 'T#000d01h30m45s000ms'.

### LD body




## Example

### POU header

This example shows how, from an input value of the data type TIME, a string STRING[9] with the format 'xxhxxmxxs' is created (only hours, minutes and seconds are output).All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | TIME | T#1h30m45s | example value |
| 1 | VAR | result_string | STRING[9] | '' | result: here '01h30m45s' |

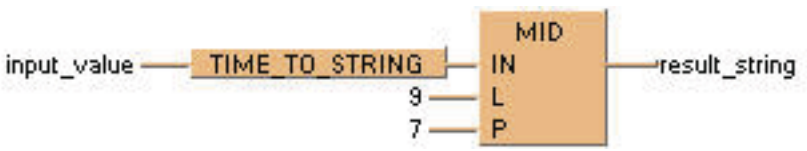
In this example, both an input variable **input_value** of the data type TIME and an output variable **result_string** of the data type STRING[9] are declared.

### POU body

In carrying out the operation in question, the standard function **MID** is attached to the function **TIME_TO_STRING**. **MID** creates a central sector in the character string from position **P** (INT value) with **L** (INT value) characters.

In the example, the INT constant 9 is entered at the **L** input of **MID**, which determines the length of the result string. The INT constant 7 at input **P** determines the position at which the central sector begins. Out of the variable **input_value** = T#1h30m45s, the STRING 'T#000d01h30m45s000ms' results from the data type conversion. The **MID** function cuts off the STRING at position 7 and yields the **result_string** '01h30m45s'.

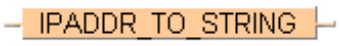
### LD body



# IPADDR_TO_STRING

IP address into STRING

This function converts a binary IP address of the data type DWORD into a STRING in IP address format.




## Remarks

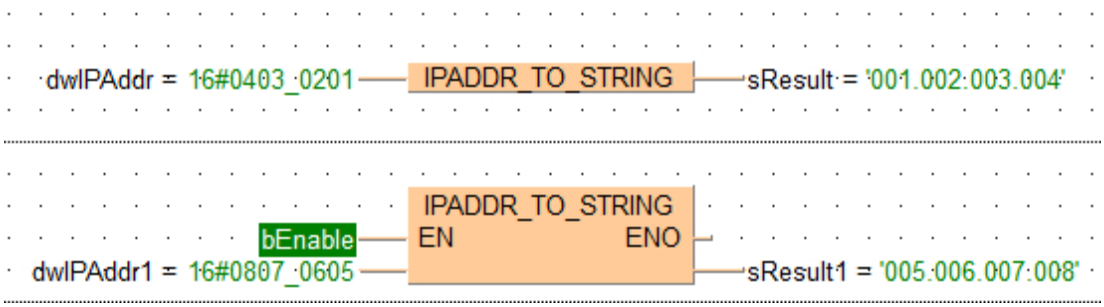
Permissible format: 'Octet1.Octet2.Octet3.Octet4', e.g.: '192.168.206.004'

Permissible characters Octets 1-4:

Decimal numbers "0“-"9“, maximal 3 positions, without leading zeros in the range 0-255

The conversion is such that the highest byte of the ET-LAN address represents the fourth octet and the lowest byte represents the first octet of the IP address. The format of the IP address corresponds to the de-facto Socket API standard.

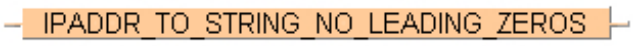
### Example



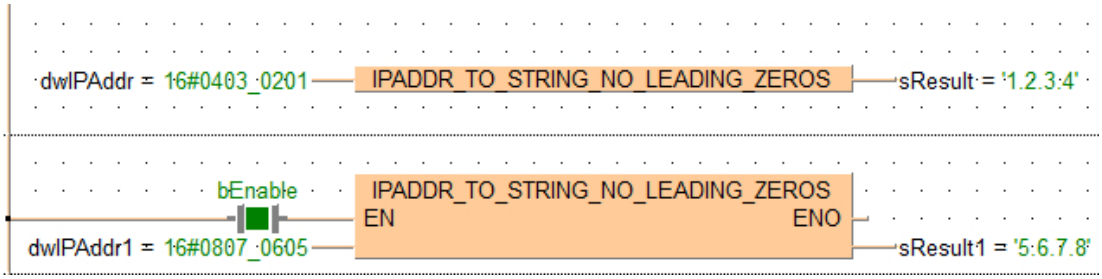
# IPADDR_TO_STRING_NO_LEADING_ZEROS

IP address into STRING

This function converts a binary IP address of the data type DWORD into a STRING in IP address format.




Example




Permissible format: 'Octet1.Octet2.Octet3.Octet4', e.g.: '192.168.206.004'

Permissible characters Octets 1-4:

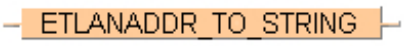
Decimal numbers "0“-"9“, maximal 3 positions, without leading zeros in the range 0-255

The conversion is such that the highest byte of the ET-LAN address represents the fourth octet and the lowest byte represents the first octet of the IP address. The format of the IP address corresponds to the de-facto Socket API standard.

# ETLANADDR_TO_STRING

ETLAN address into STRING

This function converts a binary ETLAN address of the data type DWORD into a STRING in ETLAN address format.




## Remarks

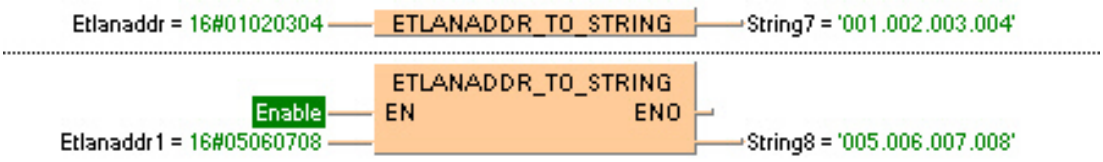
Permissible format: 'Octet1.Octet2.Octet3.Octet4', e.g.: '192.168.206.004'

Permissible characters Octets 1-4:

Decimal numbers "0“-"9“, maximal 3 positions, with leading zeros in the range 0-255

The conversion is such that the highest byte of the ET-LAN address represents the first octet and lowest byte of the IP address the fourth octet. This format for ET-LAN addresses is used, for example, by the FP Serie's ET-LAN modules.

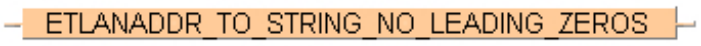
### Example



# ETLANADDR_TO_STRING_NO_LEADING_ZEROS

ETLAN address into STRING

This function converts a binary ETLAN address of the data type DWORD into a string in ETLAN address format.




Permissible format:

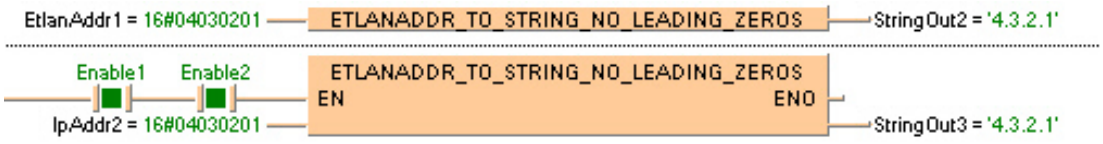
'Octet1.Octet2.Octet3.Octet4', e.g.: '192.168.206.4'

Permissible characters:

| | |
|---|---|
| Octets 1-4 | Decimal numbers "0“-"9“, maximal 3 positions, without leading zeros in the range 0-255 |

The conversion is such that the highest byte of the ETLAN address represents the first octet and lowest byte of the IP address the fourth octet. This format for ETLAN addresses is used, for example, by the FP Serie's ET-LAN modules.

### Example

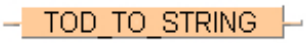


# TOD_TO_STRING

TIME_OF_DAY into STRING

**TOD_TO_STRING** converts a value of the data type TIME_OF_DAY into a value of the data type STRING[8].

The range for the input time of day is from TOD#00:00:00 to TOD#23:59:59.




## Parameters

Input

**Unnamed input** (TIME_OF_DAY)

Input data type

Output

**Unnamed output** (STRING)

Conversion resultSTRING[8]

## Remarks

All character spaces in the result string will be filled.

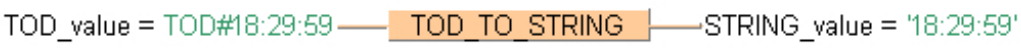
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | TOD_value | TIME_OF_DAY | TOD#18:29:59 |
| 1 | VAR | STRING_value | STRING[19] | '' |

### LD body



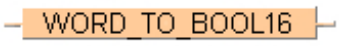
## 12.23 Conversion to Special Data Types

Special data types (only used in conversion instructions)

# WORD_TO_BOOL16

WORD into BOOL16

This function copies data of the data type WORD at the input to an array with 16 elements of the data type BOOL at the output.




## Parameters

Input

**Unnamed input** (WORD)

Input data type

Output

**Unnamed output** (ARRAY of BOOL)

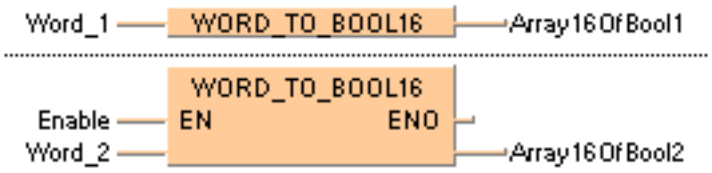
Conversion result ARRAY with 16 elements

### Example

POU header:

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Enable | BOOL | FALSE |
| 1 | VAR | Word_1 | WORD | 0 |
| 2 | VAR | Word_2 | WORD | 0 |
| 3 | VAR | Array16OfBool1 | ARRAY [0..15] OF BOOL | [16(FALSE)] |
| 4 | VAR | Array16OfBool2 | ARRAY [0..15] OF BOOL | [16(FALSE)] |

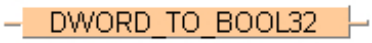
Body with and without EN/ENO:



# DWORD_TO_BOOL32

DOUBLE WORD into BOOL32

This function copies data of the data type DWORD at the input to an array with 32 elements of the data type BOOL at the output.




## Parameters

Input

**Unnamed input** (WORD)

Input data type

Output

**Unnamed output** (ARRAY of BOOL)

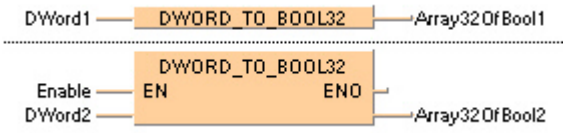
Conversion result ARRAY with 32 elements

### Example

POU header:

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Enable | BOOL | FALSE |
| 1 | VAR | Array32OfBool1 | ARRAY [0..31 OF BOOL | [FALSE] |
| 2 | VAR | Array32OfBool2 | ARRAY [0..31 OF BOOL | |
| 3 | VAR | DWord1 | DWORD | 0 |
| 4 | VAR | DWord2 | DWORD | 0 |

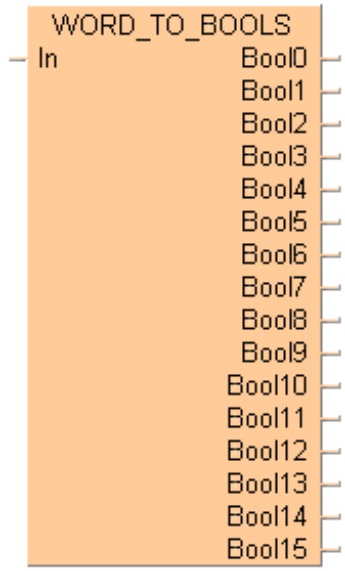
Body with and without EN/ENO:



# WORD_TO_BOOLS

WORD into 16 variables of the data type BOOL

This function converts a value of the data type WORD bit-wise to 16 values of the data type BOOL.




## Parameters

Input

**Unnamed input** (WORD)

Input data type

Output

**BOOL0 ... BOOL15** (BOOL)

Conversion result 16 output variables of the data type BOOL

## Remarks

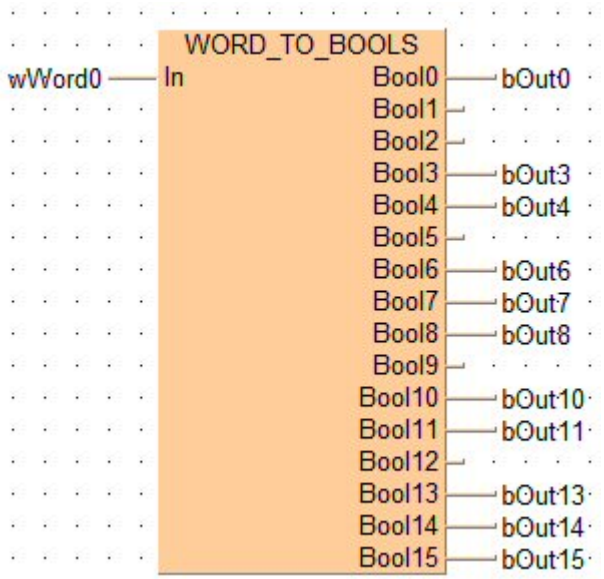
The outputs Bool0...Bool15 need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Program code is only generated for those outputs that are truly used.

### Example

POU header:

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Word0 | WORD | 0 |
| 1 | VAR | Bool0 | BOOL | FALSE |
| 2 | VAR | Bool1 | BOOL | FALSE |
| 3 | VAR | Bool2 | BOOL | FALSE |
| 4 | VAR | Bool3 | BOOL | FALSE |
| 5 | VAR | Bool4 | BOOL | FALSE |
| 6 | VAR | Bool5 | BOOL | FALSE |
| 7 | VAR | Bool6 | BOOL | FALSE |
| 8 | VAR | Bool7 | BOOL | FALSE |
| 9 | VAR | Bool8 | BOOL | FALSE |
| 10 | VAR | Bool9 | BOOL | FALSE |
| 11 | VAR | Bool10 | BOOL | FALSE |
| 12 | VAR | Bool11 | BOOL | FALSE |
| 13 | VAR | Bool12 | BOOL | FALSE |
| 14 | VAR | Bool13 | BOOL | FALSE |
| 15 | VAR | Bool14 | BOOL | FALSE |
| 16 | VAR | Bool15 | BOOL | FALSE |

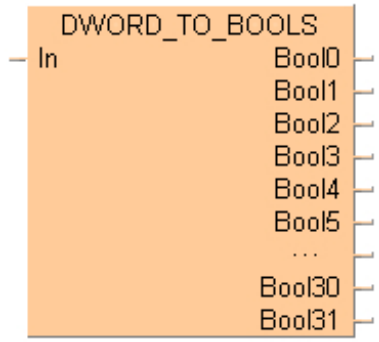
Body:



# DWORD_TO_BOOLS

DOUBLE WORD into 32 variables of the data type BOOL

This function converts a values of the data type DWORD bit-wise to 32 values of the data type BOOL.




## Parameters

Input

**Unnamed input** (WORD)

Input data type

Output

**BOOL0 ... BOOL31** (BOOL)

Conversion result 32 output variables of the data type BOOL

## Remarks

The outputs Bool0...Bool31 need not be allocated in LD or FBD, or used explicitly in the ST editor's formal list of parameters. Program code is only generated for those outputs that are truly used.

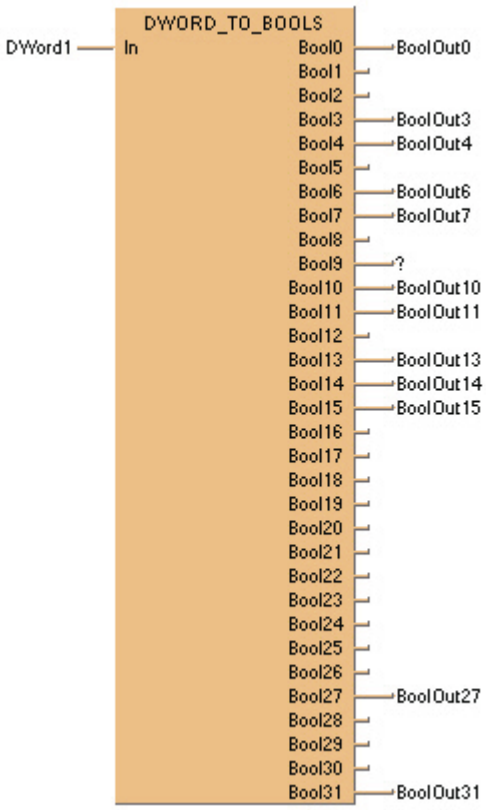
## Example

POU header:

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | dWord1 | DWORD | 0 |
| 1 | VAR | Bool0 | BOOL | FALSE |
| 2 | VAR | Bool1 | BOOL | FALSE |
| 3 | VAR | Bool2 | BOOL | FALSE |
| 4 | VAR | Bool3 | BOOL | FALSE |
| 5 | VAR | Bool4 | BOOL | FALSE |
| 6 | VAR | Bool5 | BOOL | FALSE |
| 7 | VAR | Bool6 | BOOL | FALSE |
| 8 | VAR | Bool7 | BOOL | FALSE |

etc. to Bool31

Body:



# INT_TO_BCD_WORD

INTEGER into binary coded WORD value

**INT_TO_BCD_WORD** converts a binary value of the data type INT into a binary coded decimal integer (BCD) value of the type WORD in order to be able to output BCD values in word format.


## Parameters

Input

**Unnamed input** (INT)

Input data type

Output

**Unnamed output** (BCD_WORD)

Conversion result

## Remarks

Since the output variable is of the type WORD and is therefore comprised of 16 bits, the value for the input variable is limited to 4 digits and must be between 0 and 9999.

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | BCD_value_16bit | WORD | 0 |
| 1 | VAR | INT_value | INT | 0 |

This example uses variables. You can also use a constant for the input variable.

## POU body

**INT_value** of the data type INTEGER is converted into a BCD value of the data type WORD. The converted value is written into **BCD_value_16bit**.

### LD body

# DINT_TO_BCD_DWORD

DOUBLE INTEGER into BCD DOUBLE WORD

**DINT_TO_BCD_DWORD** converts a value of the data type DINT into a BCD value of the data type DWORD.


## Parameters

Input

**Unnamed input** (DINT)

Input data type

Output

**Unnamed output** (BCD_DWORD)

Conversion result

## Remarks

The value for the input variable should be between 0 and 999,999,999.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DINT_value | DINT | 0 |
| 1 | VAR | BCD_value_32bit | DWORD | 0 |

This example uses variables. You can also use a constant for the input variable.

### POU body

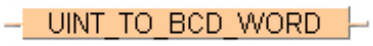
**DINT_value** of the data type DOUBLE INTEGER is converted into a BCD value of the data type DOUBLE WORD. The converted value is written to **BCD_value_32bit**.

### LD body

# UINT_TO_BCD_WORD

Unsigned INTEGER into BCD value of WORD

**UINT_TO_BCD_WORD** converts a value of the data type Unsigned INTEGER into a BCD value of the data type WORD.




## Parameters

Input

**Unnamed input** (UINT)

Input data type

Output

**Unnamed output** (BCD_WORD)

Conversion result

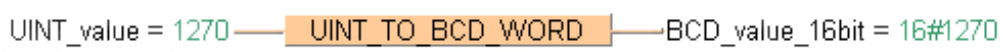
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UINT_value | UINT | 1270 |
| 1 | VAR | BCD_value_16bit | WORD | 16#0000 |

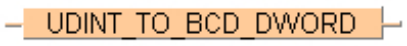
### LD body



# UDINT_TO_BCD_DWORD

Unsigned DOUBLE INTEGER into BCD DOUBLE WORD

**UDINT_TO_BCD_DWORD** converts a value of the data type Unsigned DOUBLE INTEGER into a BCD value of the data type DWORD.




## Parameters

Input

**Unnamed input** (UDINT)

Input data type

Output

**Unnamed output** (BCD_DWORD)

Conversion result

## Remarks

The value for the input variable should be between 0 and 999,999,999.

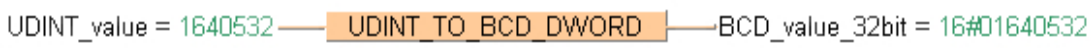
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | UDINT_value | UDINT | 16#190854 |
| 1 | VAR | BCD_value_32bit | DWORD | |

### LD body

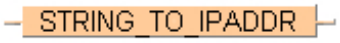


# STRING_TO_IPADDR

STRING into IP address

This function converts a STRING in IP address format into a value of the data type DWORD.

Thereby the attached string is first converted into a value of the data type STRING[32]. Finally this is converted into a value of the data type DWORD via a sub-programm of approx. 330 steps that is also used in the functions **STRING_TO_IPADDR** and **STRING_TO_ETLANADDR**.




## Parameters

Input

**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (DWORD)

Conversion result

## Remarks

- The analysis ends with the first non-decimal number after the 4th octet or in case of a format error.
- If the format is wrong the result is 0.
- The conversion is such that the first octet represents the lowest byte of the IP address and the fourth octet the highest byte of the ET-LAN address. The format corresponds to the standard format as used in "Standard Socket Application Interfaces", for example.

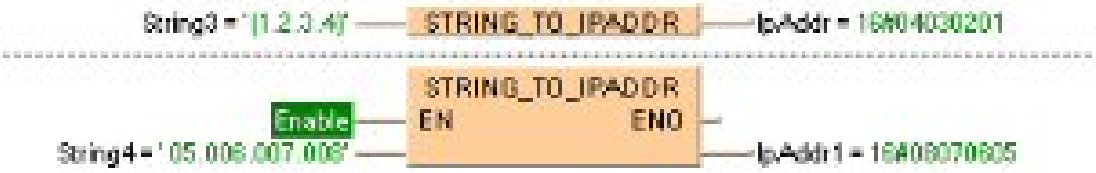
Permissible format: '[Space]Octet1.Octet2.Octet3.Octet4[Space]', e.g.: ' [192.168.206.4] '

Permissible characters Octets 1-4:

Decimal numbers "0“-"9“, maximal 3 positions, with or without leading zeros in the range 0-255

Space: All characters except for decimal numbers

**Example**


String3 = '1.2.3.4'
STRING_TO_IPADDR
IpAddr = 16#04030201
STRING_TO_IPADDR
Enable
EN
ENO
String4 = '05.006.007.008'
IpAddr1 = 16#08070605

# STRING_TO_IPADDR_STEPSAVER

STRING (IP address format 00a.0bb.0cc.ddd) into DWORD

This function converts a STRING in IP address format into a value of the data type DWORD.


## Parameters

Input

**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (DWORD)

Conversion result

## Remarks

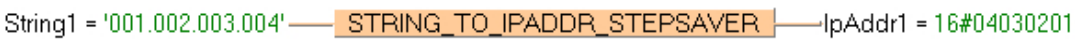
- If the format is wrong the result is 0.
- The conversion is such that the first octet of the IP address represents the lowest byte and the fourth octet represents the highest byte of the ET-LAN address. The format of the IP address corresponds to the de-facto Socket API standard.
- The function uses for approx. 50 steps of generated code the basic instructionF76_A2BIN. The instruction expects that each octet consists of three characters with leading zeros. Otherwise the PLC delivers an operation error.

Permissible format: 'Octet1.Octet2.Octet3.Octet4[Space]', e.g.: ' [192.168.206.4] '

Permissible characters Octets 1-4:

Decimal numbers "0“-"9“, maximal 3 positions, with or without leading zeros in the range 0-255

### Example

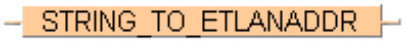


# STRING_TO_ETLANADDR

STRING into ETLAN address

This function converts a STRING in IP address format into a value of the data type DWORD.

Thereby the attached string is first converted into a value of the data type STRING[32]. Finally this is converted into a value of the data type DWORD via a sub-program of approx. 330 steps that is also used in the functions **STRING_TO_IPADDR** and **STRING_TO_ETLANADDR**.




## Parameters

Input

**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (DWORD)

Conversion result

## Remarks

- The analysis ends with the first non-decimal number after the 4th octet or in case of a format error.
- If the format is wrong the result is 0.
- The conversion is such that the highest byte of the ET-LAN address represents the first octet and lowest byte of the IP address the fourth octet. This format for ET-LAN addresses is used, for example, by the FP Serie's ET-LAN modules.

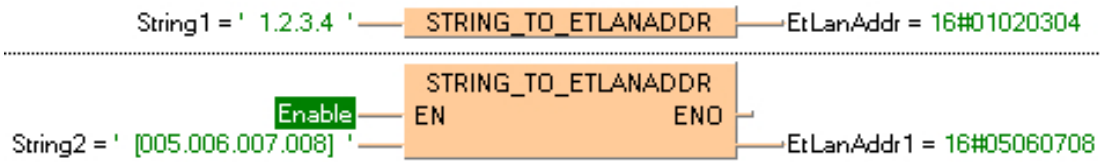
Permissible format: '[Space]Octet1.Octet2.Octet3.Octet4[Space]', e.g.: ' [192.168.206.4] '

Permissible characters Octets 1-4:

Decimal numbers "0“-"9“, maximal 3 positions, with or without leading zeros in the range 0-255

Space: All characters except for decimal numbers

**Example with and without EN/ENO:**



# STRING_TO_ETLANADDR_STEPSAVER

STRING (IP address format 00a.0bb.0cc.ddd) into ETLAN address

This function converts a STRING in IP address format into a value of the data type DWORD.


## Parameters

Input

**Unnamed input** (STRING)

Input data type

Output

**Unnamed output** (DWORD)

Conversion result

## Remarks

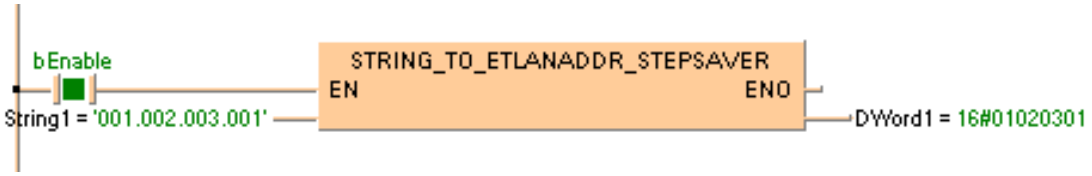
- If the format is wrong the result is 0.
- The conversion is such that the highest byte of the ET-LAN address represents the first octet and lowest byte of the IP address the fourth octet. This format for ET-LAN addresses is used, for example, by the FP Serie's ET-LAN modules.
- The function uses for approx. 50 steps of generated code the basic instructionF76_A2BIN. The instruction expects that each octet consists of three characters with leading zeros. Otherwise the PLC delivers an operation error.

Permissible format: 'Octet1.Octet2.Octet3.Octet4[Space]', e.g.: ' [192.168.206.4] '

Permissible characters Octets 1-4:

Decimal numbers "0“-"9“, maximal 3 positions, with or without leading zeros in the range 0-255

### Example



## 12.24 FP instructions

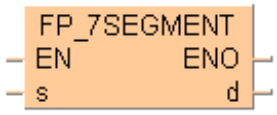
Tip

Advantages of FP instructions

# FP_7SEGMENT

7-segment decode

This FP instruction converts the data or constant specified by **s** to 4-digit data for 7-segment indication if the trigger **EN** is TRUE. The converted data is stored in the area starting at **d**. 7-segment indication requires 8 bits (1 byte) to represent 1 digit.




## Parameters

Input

**s** (WORD)

Data or constant

Output

**d** (DWORD)

Converted data or constant

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the result is out of range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the result is out of range

## Example

## POU header

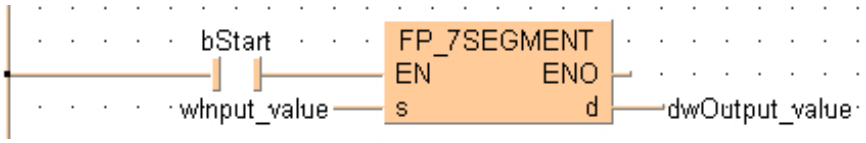
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 1 | VAR | bStart | BOOL | FALSE | activates the function |
| 2 | VAR | wInput_value | WORD | 16#A731 | |
| 3 | VAR | dwOutput_value | DWORD | 0 | result after 0->1 leading edge from start: 16#77274F06 |
| 4 | VAR | | | | |

## POU body

When the variable **bStart** is set to TRUE, the function is carried out.

## LD body

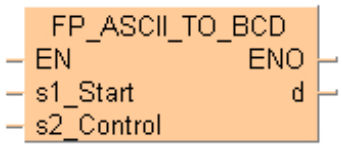


# FP_ASCII_TO_BCD

ASCII -> BCD conversion

This FP instruction converts the hexadecimal ASCII codes at **s1_Start** to BCD characters if the trigger **EN** is TRUE. **s2_Control** specifies the number of bytes to be converted and the conversion direction.

The result is stored in **d**.




## Parameters

Input

**s1_Start** (WORD, INT, UINT)

Starting address

**s2_Control** (WORD)

Number of bytes and conversion direction

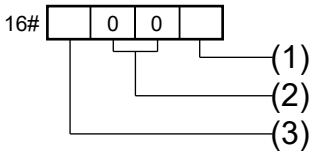
Output

**d** (WORD, DWORD)

Converted bytes

## Remarks

- Specifying the control code **s2_Control**



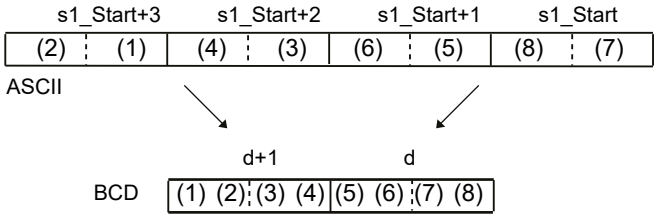

(1) data size to be converted:
16-bit: 1–2 bytes to be converted
32-bit: 1–4 bytes to be converted

(2) always 0

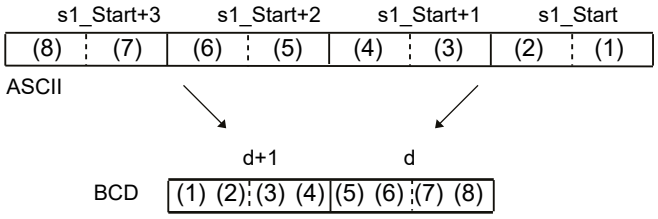
(3) conversion direction:
0: forward
1: reverse

- The two characters that make up one byte are interchanged when stored. Two bytes are converted as one segment of data.

– Forward direction:




– Reverse direction:




- ASCII HEX codes to express BCD characters:

| BCD character | ASCII HEX code |
|---|---|
| 0123456789 | 16#30 16#31 16#32 16#33 16#34 16#35 16#36 16#37 16#38 16#39 |

ExampleSigned/unsigned 16-bit data, forward direction, 4 characters

| | Offset | ASCII codes | BCD equivalent | | | Offset | Converted BCD characters |
|---|---|---|---|---|---|---|---|
| **s1_Start** | 0 | 16#3231 | 21 | ⇒ | **d** | 0 | 16#3412 |
| | 1 | 16#3433 | 3 | | | 1 | |
| **s2_Control** | | 16#0004 | | | | | |

Example16-bit data, reverse direction, 4 characters

| | Offset | ASCII codes | BCD equivalent | | | Offset | Converted BCD characters |
|---|---|---|---|---|---|---|---|
| **s1_Start** | 0 | 16#3231 | 21 | ⇒ | **d** | 0 | 16#1234 |
| | 1 | 16#3433 | 43 | | | 1 | |
| **s2_Control** | | 16#1004 | | | | | |

Example32-bit data, forward direction, 8 characters

| | Offset | ASCII codes | BCD equivalent | | | Offset | Converted BCD characters |
|---|---|---|---|---|---|---|---|
| **s1_Start** | 0 | 16#3231 | 21 | ⇒ | **d** | 0 | 16#3412 |
| | 1 | 16#3433 | 3 | | | 1 | 16#7856 |
| | 2 | 16#3635 | 65 | | | 2 | |
| | 3 | 16#3837 | 87 | | | 3 | |
| **s2_Control** | | 16#0008 | | | | | |

Example32-bit data, reverse direction, 8 characters

| | Offset | ASCII codes | BCD equivalent | | | Offset | Converted BCD characters |
|---|---|---|---|---|---|---|---|
| **s1_Start** | 0 | 16#3231 | 21 | ⇒ | **d** | 0 | 16#5678 |
| | 1 | 16#3433 | 43 | | | 1 | 16#1234 |
| | 2 | 16#3635 | 65 | | | 2 | |
| | 3 | 16#3837 | 87 | | | 3 | |
| **s2_Control** | | 16#1008 | | | | | |

Example32-bit data, reverse direction, 7 characters

If an odd number of characters is being converted, 0 will be entered for bit position 0–3 of the last converted byte if the conversion direction is "forward". If the conversion direction is "reverse", 0 will be entered for bit position 4–7:

| | Offset | ASCII codes | BCD equivalent | | | Offset | Converted BCD characters |
|---|---|---|---|---|---|---|---|
| **s1_Start** | 0 | 16#3231 | 21 | ⇒ | **d** | 0 | 16#4567 |
| | 1 | 16#3433 | 43 | | | 1 | 16#0123 |
| | 2 | 16#3635 | 65 | | | 2 | |
| | 3 | 16#3837 | 87 | | | 3 | |
| **s2_Control** | | 16#1007 | | | | | |

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if ASCII characters other than 16#30–16#39 are specified
- if the number of bytes specified by **s2_Control** is greater than the area specified by **s1_Start**
- if the conversion result is greater than the data area specified by **d**
- if **s2_Control** = 0
- if the conversion direction is out of range
- if the number of bytes specified in **s2_Control** is greater than 8

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if ASCII characters other than 16#30–16#39 are specified
- if the number of bytes specified by **s2_Control** is greater than the area specified by **s1_Start**
- if the conversion result is greater than the data area specified by **d**

- if **s2_Control** = 0
- if the conversion direction is out of range
- if the number of bytes specified in **s2_Control** is greater than 8

## Example

### POU header

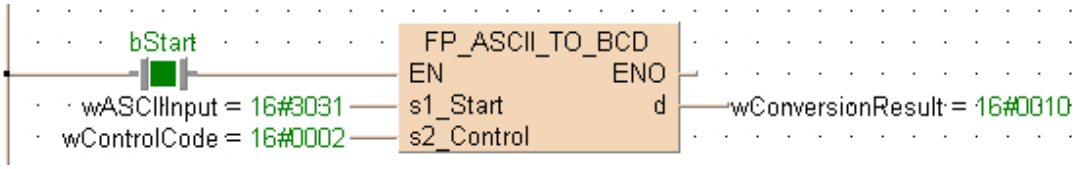
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | wASCIIInput | WORD | 16#3031 | |
| 2 | VAR | wControlCode | WORD | 16#2 | specifies the operation: |
| 3 | VAR | wConversionResult | WORD | 0 | result after a 0->1 leading |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out. Two characters of the ASCII input value are converted in forward direction

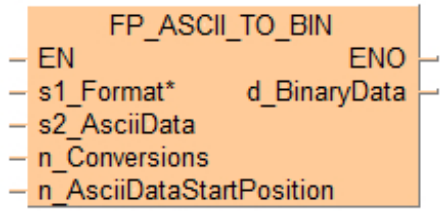
### LD body



# FP_ASCII_TO_BIN

ASCII -> binary conversion

This FP instruction converts ASCII code stored in the area specified by **s2_AsciiData** to 16-bit/32-bit binary data. The conversion method is specified by **s1_Format**. The converted result is stored in the area specified by **d_BinaryData**.




Note

The FP instructions **FP_ASCII_TO_BIN** and **FP_ASCII_CHECK** do not differentiate between upper and lower case.

## Parameters

Input

**s1_Format** (STRING)

Control and format string set in inverted commas

**s2_AsciiData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Starting address for storing ASCII data

**n_Conversions** (WORD, INT, UINT)

Quantity of numbers to be converted: 0–65535

**n_AsciiDataStartPosition** (INT)

Starting position in ASCII data: 0–255

Output

**d_BinaryData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Starting address for storing binary data

### Explanation of each digit of the control code s1_Format:

| Position in the control code | Description | | s1_Format | 16-bit PLCs | 32-bit PLCs |
|---|---|---|---|---|---|
| '+%08lX' | Conversion direction | | | | |
| | **+** | forward (only for hexadecimal numbers with specifier **x**, **X**) | '+%4X' | ● | ● |
| | | if plus sign is omitted: reverse (default setting) | '%4X' | ● | ● |
| '+%08lX' | **%** | format string specifier (mandatory) | | ● | ● |
| '+%08lX' | Padding format | | | | |
| | **0** | fill with zeros | '%06x' | - | ● |
| | **+** | add a plus sign | '%+4d' | - | ● |
| | **-** | left alignment | '%-6d' | - | ● |
| | **␣** | (space) add a space instead of plus sign | '%␣4d' | - | ● |
| | # | insert 0x for hexadecimal numbers | '%#4X' | - | ● |
| | | append always a decimal point for real number | '%#8.0f' | - | ● |
| '+%08lX' | **8** | width of the ASCII data element | '%08d' | ● | ● |
| | no width<br>• **FP_BIN_TO_ASCII**: required minimum width is assumed<br>• **FP_ASCII_TO_BIN** and **FP_ASCII_CHECK**: comma separator is required | | '%d,' | - | ● |
| '+%#8.5lx' | Precision after decimal point | | | | |
| | **.5** | any digit after decimal point | '%8.5f' | - | ● |
| '+%08lX' | **l** | double length specifier<br>e.g. specifier i with l = li -> DINT | '+%4ld' | ● | ● |
| '+%08lX' | Format specifier | | | | |
| | **i** | INT | '%10i' | ● | ● |
| | **u** | UINT | '%10u' | - | ● |
| | **d** | INT | '%6d' | ● | ● |
| | **x** | hexadecimal lower case | '+%4x' | ● | ● |
| | **X** | hexadecimal upper case | '+%4X' | ● | ● |
| | **b** | BCD | '+%5b' | - | ● |
| | **f** | floating-point number | '+%-6.2f' | - | ● |
| | **e** | exponential 1.23e10 | '+%9.3e' | - | ● |
| | **E** | exponential upper case 1.23E10 | '+%9.3E' | - | ● |

| Position in the control code | Description | | s1_Format | 16-bit PLCs | 32-bit PLCs |
|---|---|---|---|---|---|
| | **g** | floating or exponential | '+%12g' | - | ● |
| | **G** | floating or exponential upper case | '+%9.3G' | - | ● |
| '%8dPANA' | • For **FP_BIN_TO_ASCII**: any string can be appended to the conversion e.g. 'PANA'<br>• For **FP_ASCII_TO_BIN** and **FP_ASCII_CHECK**: only comma is allowed | | '+%8dPANA' | - | ● |

● available on 16-bit/32-bit PLCs

### Examples of control string s1_Format

- Example: '+%4X' converts an ASCII value with a width of four characters in upper case in forward direction (valid for 16-bit/32-bit PLCs)

| ASCII data | Conversion result in binary data |
|---|---|
| '␣12A' | 16#12A (displayed in hexadecimal format) |

- Example: '%4X' converts an ASCII value with a width of four characters in upper case in reverse direction (valid for 16-bit/32-bit PLCs)

| ASCII data | Conversion result in binary data |
|---|---|
| '␣B2A' | 16#B2A (displayed in hexadecimal format) |

- Example: '%06d' converts an ASCII value with a width of six characters, decimal value with three leading zeros (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '000100' | 16#100 (displayed in hexadecimal format) |

- Example: '%+4d' converts an ASCII value with a width of four characters, decimal value, + sign added (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '+100␣␣' | 100 (displayed in decimal format) |

- Example: '%-6d' converts an ASCII value with a width of six characters, left aligned (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '100␣␣␣' | 100 (displayed in decimal format) |

- Example: `'%␣4d'` converts an ASCII value with a width of four characters with one leading space (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '␣100' | 100 (displayed in decimal format) |

- Example: `'%#8.0f'` converts an ASCII value with a width of eight characters, floating-point number, four leading spaces (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '␣␣␣␣123.' | 123.45678 (displayed in decimal format) |

- Example: `'%8.3f'` converts an ASCII value with a width of eight characters, three characters precision after decimal point (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '␣123.456' | 123.45599 (displayed in decimal format) |

- Example: `'+%10u'` converts an ASCII value with a width of 10 characters, seven leading spaces, unsigned integer (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '␣␣␣␣␣␣␣100' | -100 (displayed in decimal format) |

- Example: `'%06d'` converts an ASCII value with a width of six characters with three leading spaces (valid for 16-bit/32-bit PLCs)

| ASCII data | Conversion result in binary data |
|---|---|
| '␣␣␣100' | 100 (displayed in decimal format) |

Permissible range for the value before the specifier:

1–15 before specifiers **d**, **ld**, **i**, **li**

1–4 before specifier **X**

1–8 before specifier **lX**

if no width is specified, comma is appended e.g. `'%d,'` (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '100,' | 100 (displayed in decimal format) |

  - **FP_BIN_TO_ASCII**: required minimum width is assumed
  - **FP_ASCII_TO_BIN** and **FP_ASCII_CHECK**: comma separator is required

- Example: `'+%4ld'` converts an ASCII value with a width of four characters, requires DINT or DWORD for the converted result (valid for 16-bit/32-bit PLCs)

| ASCII data | Conversion result in binary data |
|---|---|
| '␣100' | 100 (displayed in decimal format) |

- Example: `'+%6i'` converts an ASCII value with a width of six characters, three leading spaces, signed integer in forward direction (valid for 16-bit/32-bit PLCs)

| ASCII data | Conversion result in binary data |
|---|---|
| '␣␣␣-100' | -100 (displayed in decimal format) |

- Example: `'+%6d'` converts an ASCII value with a width of six characters with three leading zeros in forward direction (valid for 16-bit/32-bit PLCs)

| ASCII data | Conversion result in binary data |
|---|---|
| '000100' | 100 (displayed in decimal format) |

- Example: `'+%4X'` converts an ASCII value with a width of four characters, hexadecimal number in upper case in forward direction (valid for 16-bit/32-bit PLCs)

| ASCII data | Conversion result in binary data |
|---|---|
| '␣12A' | 16#12A (displayed in hexadecimal format) |

- Example: `'+%5b'` converts an ASCII value with a width of five characters, BCD data (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '␣123' | 16#123 (displayed in hexadecimal format) |

- Example: `'+%-6.2f'` converts an ASCII value with a width of six characters, left aligned, precision two digits after decimal point (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '1.23␣' | 1.2345 (displayed in decimal format) |

- Example: `'+%9.3e'` converts an ASCII value with a width of 9 characters, precision 3 digits after decimal point, exponential lower case (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '1.235e+03' | 1234.5678 (displayed in decimal format) |

- Example: `'1.235E+03'` converts an ASCII value with a width of 9 characters, precision 3 digits after decimal point, exponential upper case (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '1.235E+03' | 1234.5678 (displayed in decimal format) |

- Example: `'+%12g'` converts an ASCII value with a width of 12 characters, floating-point number (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '␣␣␣␣␣1234.57' | 1234.5678 (displayed in decimal format) |

- Example: `'+%9.3G'` converts an ASCII value with a width of nine characters, precision three characters after decimal point, floating-point number, exponential upper case (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '␣␣␣1.E+03' | 1234 (displayed in decimal format) |

- For **FP_BIN_TO_ASCII**: any string can be appended to the conversion
  Example: `'+%8dPANA'` converts an ASCII value with a width of eight characters, decimal number, 'PANA' is appended (valid for 32-bit PLCs only)

| ASCII data | Conversion result in binary data |
|---|---|
| '␣␣␣␣␣100PANA' | 100 (displayed in decimal format) |

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if there is an error in the control string specified by **sFormat**.

- if forward direction (+) is specified in **sFormat** when the format is decimal.
- if the number of ASCII characters per converted unit specified by **n_Conversions** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specified by **s1_Format**.
- if 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_Conversions**.
- if the number of 16- or 32-bit decimal numbers to be converted specified by **n_Conversions** exceeds the area for storing ASCII data.
- if the converted result exceeds the area.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if there is an error in the control string specified by **sFormat**.
- if forward direction (+) is specified in **sFormat** when the format is decimal.
- if the number of ASCII characters per converted unit specified by **n_Conversions** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specified by **s1_Format**.
- if 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_Conversions**.
- if the number of 16- or 32-bit decimal numbers to be converted specified by **n_Conversions** exceeds the area for storing ASCII data.
- if the converted result exceeds the area.

## Example

### POU header

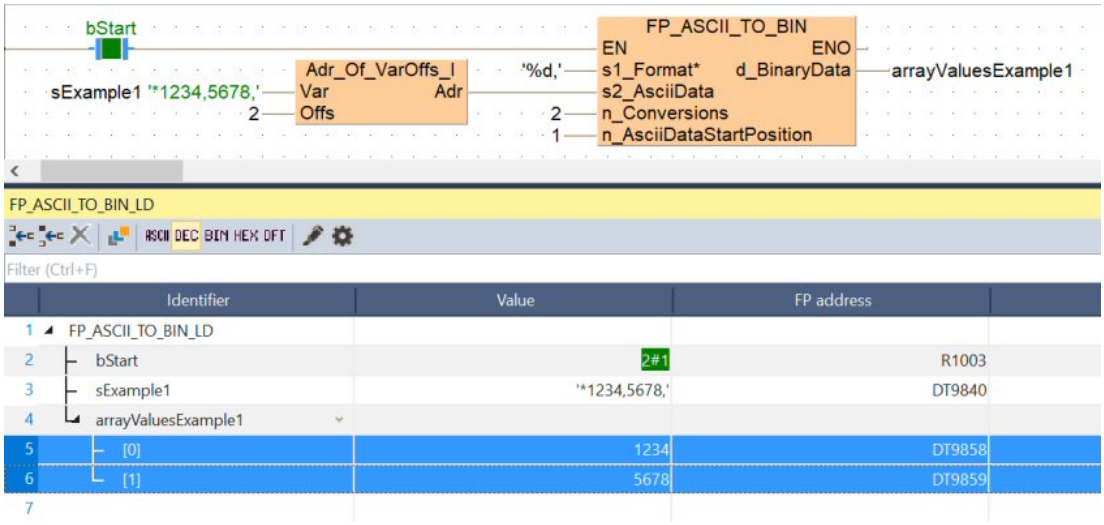
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | sExample1 | STRING[32] | '*1234,5678,' |
| 2 | VAR | arrayValuesExample1 | ARRAY [0..1] OF UINT | [2(0)] |

### POU body

When **bStart** is set to TRUE, the instruction is carried out. It converts 2 x 4 decimal ASCII characters to binary data. Offset = 1 ASCII character (8-bit).

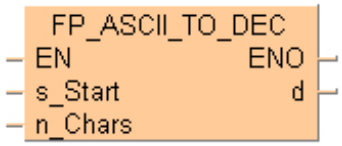
## LD body


bStart
FP_ASCII_TO_BIN
EN
ENO
Adr_Of_VarOffs_I
sExample1 '*1234,5678,'
Var
Adr
2
Offs
'%d,'
s1_Format*
d_BinaryData
arrayValuesExample1
s2_AsciiData
2
n_Conversions
1
n_AsciiDataStartPosition
FP_ASCII_TO_BIN_LD
ASCII DEC BIN HEX OCT
Filter (Ctrl+F)
Identifier
Value
FP address
1 FP_ASCII_TO_BIN_LD
2 bStart 2#1 R1003
3 sExample1 '*1234,5678,' DT9840
4 arrayValuesExample1
5 [0] 1234 DT9858
6 [1] 5678 DT9859
7

# FP_ASCII_TO_DEC

ASCII -> decimal conversion

This FP instruction converts the hexadecimal ASCII codes beginning at **s_Start** to decimal numbers if the trigger **EN** is TRUE. **n_Chars** specifies the number of bytes to be converted.The result is stored in **d**.




## Parameters

Input

**s_Start** (WORD, INT, UINT)

Starting address

**n_Chars** (WORD, INT, UINT)

Number of bytes

Output

**d** (INT, DINT, UINT, UDINT)

Converted bytes

## Remarks

ASCII HEX codes to express decimal characters:

| Decimal characters | ASCII HEX code |
|---|---|
| SPACE+-0123456789 | 16#20 16#2B 16#2D 16#30 16#31 16#32 16#33 16#34 16#35 16#36 16#37 16#38 16#39 |

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the number of bytes specified by **n_Chars** is greater than the area specified by **d**
- if the conversion result is greater than the data area specified by **d**
- if ASCII characters other than 0–9, signs (+, –) or space is specified

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the number of bytes specified by **n_Chars** is greater than the area specified by **d**
- if the conversion result is greater than the data area specified by **d**
- if ASCII characters other than 0–9, signs (+, –) or space is specified

### Example 32-bit data, unsigned

| | Offset | ASCII codes | Dec. equivalent | | | Offset | Converted decimal characters | Decimal figure |
|---|---|---|---|---|---|---|---|---|
| **s_Start** | 0 | 16#3620 | 6 | ⇒ | **d** | 0 | 16#FF9C | 65436 |
| | 1 | 16#3435 | 45 | | | 1 | | |
| | 2 | 16#3633 | 63 | | | | | |
| **n_Chars** | 0 | 16#0006 | | | | | | |

### Example 32-bit data, signed

| | Offset | ASCII codes | Dec. equivalent | | | Offset | Converted decimal characters | Decimal figure |
|---|---|---|---|---|---|---|---|---|
| **s_Start** | 0 | 16#2020 | | ⇒ | **d** | 0 | 16#FF9C | -100 |
| | 1 | 16#312D | 1 - | | | 1 | | |
| | 2 | 16#3030 | 00 | | | | | |
| **n_Chars** | 0 | 16#000A | | | | | | |

### Example 32-bit data, unsigned

| | Offset | ASCII codes | Dec. equivalent | | | Offset | Converted decimal characters | Decimal figure |
|---|---|---|---|---|---|---|---|---|
| **s_Start** | 0 | 16#3234 | 24 | ⇒ | **d** | 0 | 16#FF9C | 4294967196 |
| | 1 | 16#3439 | 49 | | | 1 | 16#FFFF | |
| | 2 | 16#3639 | 69 | | | 2 | | |
| | 3 | 16#3137 | 17 | | | | | |
| | 4 | 16#3639 | 69 | | | | | |
| **n_Chars** | 0 | 16#000A | | | | | | |

### Example 32-bit data, signed

| | Offset | ASCII codes | Dec. equivalent | | | Offset | Converted decimal characters | Decimal figure |
|---|---|---|---|---|---|---|---|---|
| **s_Start** | 0 | 16#2020 | | ⇒ | **d** | 0 | 16#FF9C | -100 |
| | 1 | 16#2020 | | | | 1 | 16#FFFF | |
| | 2 | 16#2020 | | | | | | |
| | 3 | 16#312D | | | | | | |
| | 4 | 16#3030 | | | | | | |
| **n_Chars** | 0 | 16#000A | | | | | | |

## Example

### POU header

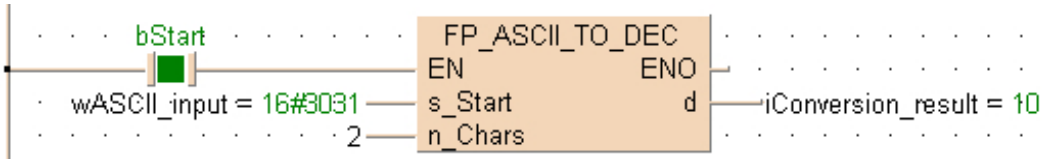
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | wASCII_input | WORD | 16#3031 | |
| 2 | VAR | iConversion_result | INT | 0 | |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

### LD body

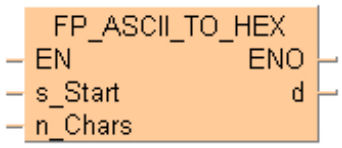


# FP_ASCII_TO_HEX

ASCII -> HEX conversion

This FP instruction converts the hexadecimal ASCII HEX codes at **s_Start** to hexadecimal characters if the trigger **EN** is TRUE. **n_Chars** specifies the number of bytes to be converted.The result is stored in **d**.

ASCII code requires 8 bits (1 byte) to express one hexadecimal character.Upon conversion, the data length will thus be half the length of the ASCII code source data.




## Parameters

Input

**s_Start** (WORD, INT, UINT)

Starting address

**n_Chars** (INT, DINT, UINT, UDINT)

Number of bytes

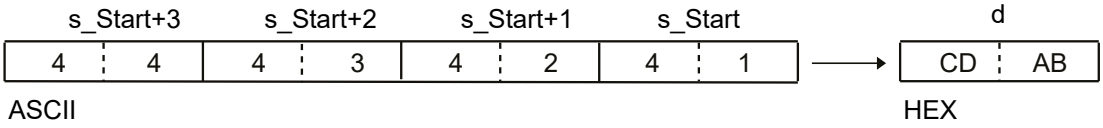
Output

**d** (WORD, DWORD)

Converted bytes

## Remarks

Two characters of ASCII code are converted into two one-byte numerical digits. In this process, higher-level characters and lower-level characters are exchanged. Four characters are converted as one segment of data.




ASCII HEX codes to express hexadecimal characters:

| Hexadecimal character | ASCII HEX code |
|---|---|
| 0123456789ABCDEF | 16#3016#31 16#32 16#33 16#34 16#35 16#36 16#37 16#38 16#39 16#41 16#42 16#43 16#44 16#45 16#46 |

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the number of bytes specified by **n_Chars** is greater than the area specified by **s_Start**
- if the conversion result is greater than the data area specified by **d**
- if **n_Chars** = 0
- if ASCII characters other than 0–F are specified

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the number of bytes specified by **n_Chars** is greater than the area specified by **s_Start**
- if the conversion result is greater than the data area specified by **d**
- if **n_Chars** = 0
- if ASCII characters other than 0–F are specified

### Example Signed/unsigned 16-bit data:

| | Offset | ASCII codes | Hex. equivalent | | | Offset | Converted hex. characters |
|---|---|---|---|---|---|---|---|
| **s1_Start** | 0 | 16#4241 | BA | ⇒ | **d** | 0 | 16#CDAB |
| | 1 | 16#4443 | DC | | | 1 | |
| **n_Chars** | 0 | 16#0004 | | | | 2 | |
| | 1 | | | | | 3 | |

### Example Signed/unsigned 32-bit data:

| | Offset | ASCII codes | Hex. equivalent | | | Offset | Converted hex. characters |
|---|---|---|---|---|---|---|---|
| **s1_Start** | 0 | 16#4241 | BA | ⇒ | **d** | 0 | 16#CDAB |
| | 1 | 16#4443 | DC | | | 1 | |
| **n_Chars** | 0 | 16#0004 | | | | 2 | |
| | 1 | 16#0000 | | | | 3 | |
| | 2 | | | | | 4 | |

### Example Signed/unsigned 16-bit data:

| | Offset | ASCII codes | Hex. equivalent | | | Offset | Converted hex. characters |
|---|---|---|---|---|---|---|---|
| **s1_Start** | 0 | 16#3231 | 21 | ⇒ | **d** | 0 | 16#3412 |
| | 1 | 16#3433 | 43 | | | 1 | 16#7856 |
| | 2 | 16#3635 | 65 | | | 2 | |
| | 3 | 16#3837 | 87 | | | 3 | |
| **n_Chars** | 0 | 16#0008 | | | | 4 | |
| | 1 | | | | | 5 | |

### Example Signed/unsigned 16-bit data, odd number of characters:

If an odd number of characters is being converted, 0 will be entered for bit position 0–3 of the last converted byte.

| | Offset | ASCII codes | Hex. equivalent | | | Offset | Converted hex. characters |
|---|---|---|---|---|---|---|---|
| **s1_Start** | 0 | 16#3231 | 21 | ⇒ | **d** | 0 | 16#3412 |
| | 1 | 16#3433 | 43 | | | 1 | 16#7056 |
| | 2 | 16#3635 | 65 | | | 2 | |
| | 3 | 16#3837 | 87 | | | 3 | |
| **n_Chars** | 0 | 16#0007 | | | | 4 | |
| | 1 | | | | | 5 | |

## Example

### POU header

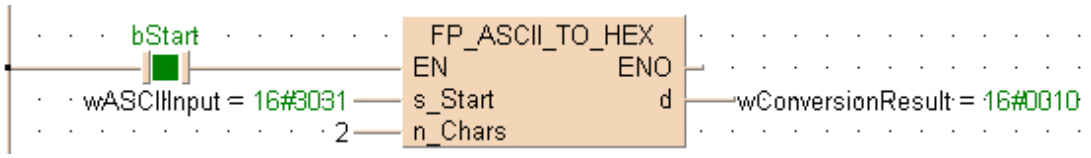
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | wASCIIInput | WORD | 16#3031 | |
| 2 | VAR | wConversionResult | WORD | 0 | |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

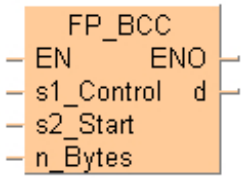
### LD body


bStart
FP_ASCII_TO_HEX
EN
ENO
wASCIIInput = 16#3031
s_Start
d
wConversionResult = 16#0010
2
n_Chars

# FP_BCC

Block check character calculation

This FP instruction calculates the block check character (BCC) according to the calculation specified by **s1_Control**. The data is specified by the start address **s2_Start** and the number of bytes **n_Bytes**. The BCC is stored in the 16-bit area specified by **d**.




## Parameters

Input

**s1_Control** (WORD, INT, UINT)

Calculation method

**s2_Start** (WORD, INT, UINT)

Starting address

**n_Bytes** (WORD, INT, UINT)

Number of bytes

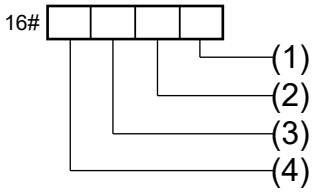
Output

**d** (WORD, INT, UINT)

Block check character

## Remarks

Specifying the control code **s1_Control**




(1) Calculation method
0: Addition (**SYS_BCC_CALCULATION_METHOD_ADD**)
1: Subtraction (**SYS_BCC_CALCULATION_METHOD_SUB**)
2: Exclusive OR (**SYS_BCC_CALCULATION_METHOD_XOR**)
16#A: CRC-16 (**SYS_BCC_CALCULATION_METHOD_CRC16**)

(2) Calculation starting position (bytes from **s2_Start**)
16#0–16#F

(3) Storage starting position (bytes from **d**)
0: Binary data (1 byte)
1: ASCII data (2 bytes)

(4) Data to be converted

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

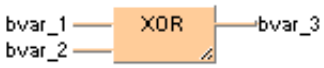
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | iBCC_Calc_Method | INT | 2 | |
| 2 | VAR | sASCII_String | STRING[32] | '%01#RCSX0000' | specifies the operation: |
| 3 | VAR | wBCC | WORD | 0 | result after a 0->1 leading |

### POU body

When **bStart** turns to TRUE, a block check character calculation is performed on **sASCII_String**. The calculation method is exclusive OR. (Use this method when large amounts of data are transmitted).

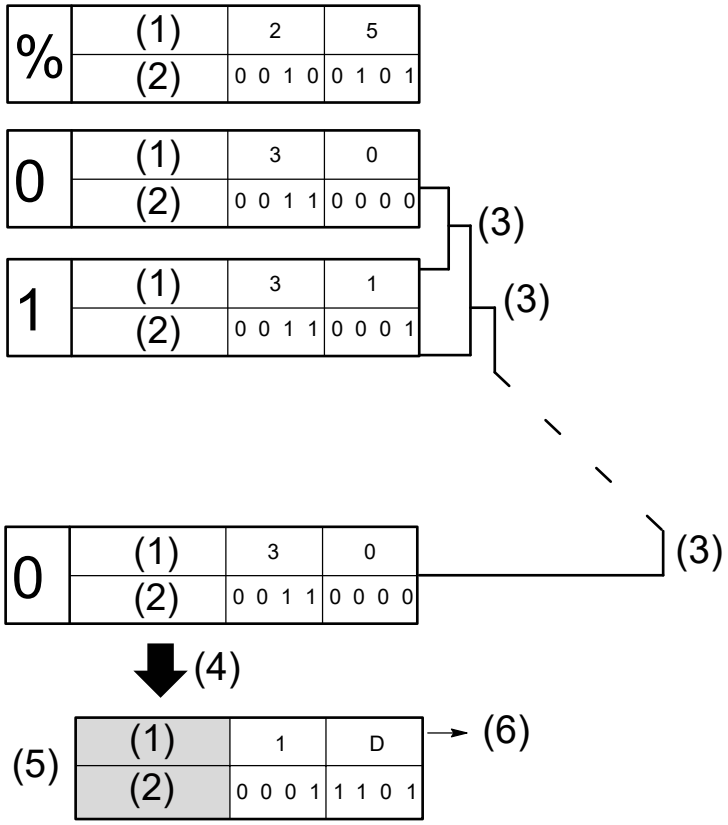
Exclusive OR operation:

| bvar_1 | bvar_2 | bvar_3 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



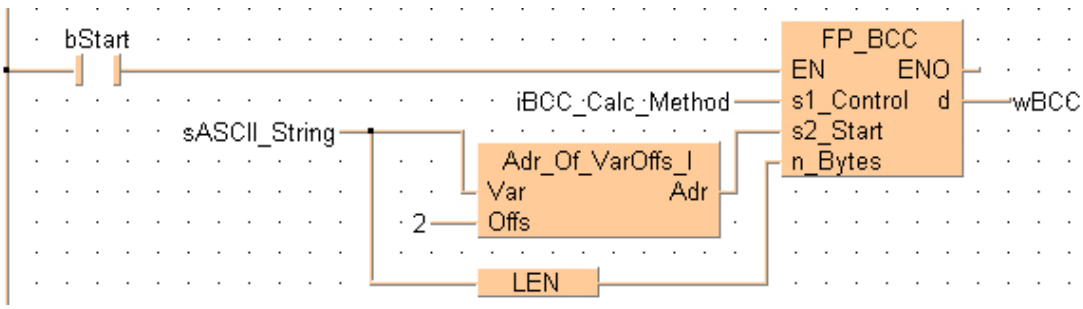

The binary codes of the first two characters are compared with each other to yield an 8-character exclusive OR operation result. This result is then compared to the binary code of the next character, and so on until the final character is reached. The last exclusive OR result is the block check character.

BCC calculation using exclusive OR operation:




(1) ASCII-HEX-code
(2) ASCII-BIN-code
(3) Exclusive ORing
(4) calculation
(5) Block Check Character (BCC)
(6) Calculation result (16#1D) is stored in **d**.

## LD body



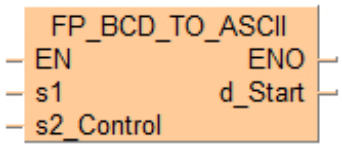

In this example, **Adr_Of_VarOffs_I** is used to create an offset of 2 words to compensate for the string's start code. By using **LEN**, the BCC calculation is performed on the entire data string.

# FP_BCD_TO_ASCII

BCD -> ASCII conversion

This FP instruction converts the BCD code at **s1** to ASCII code if the trigger **EN** is TRUE. **s2_Control** specifies the number of bytes to be converted and the conversion direction.

The result is stored in the area specified by **d_Start**. ASCII code requires 8 bits (1 byte) to express 1 BCD character. Upon conversion to ASCII, the data length will thus be twice the length of the source data.




## Parameters

Input

**s1** (WORD, DWORD)

BCD code

**s2_Control** (WORD)

Number of bytes and conversion direction

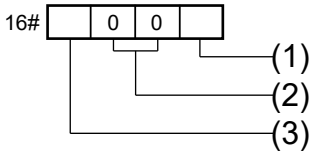
Output

**d_Start** (WORD, INT, UINT)

Starting address of the data area for the results. The size is **n_Bits** * 2 words.

## Remarks

Specifying the control code **s2_Control**



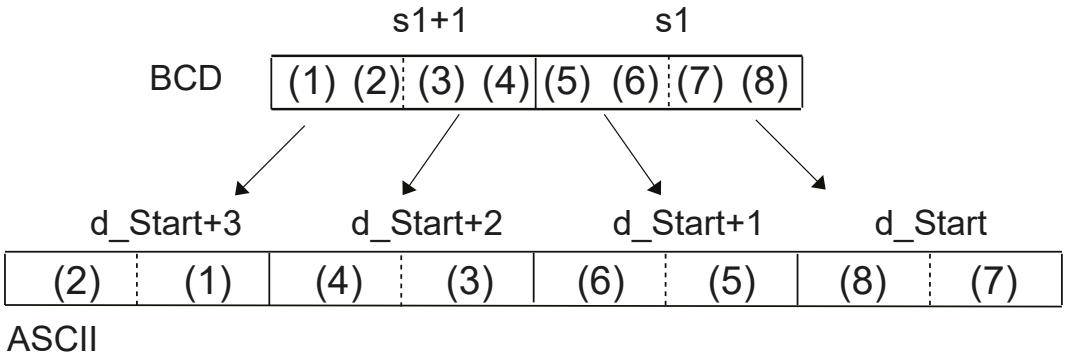

(1) data size to be converted:
16-bit: 1–2 bytes to be converted
32-bit: 1–4 bytes to be converted

(2) always 0

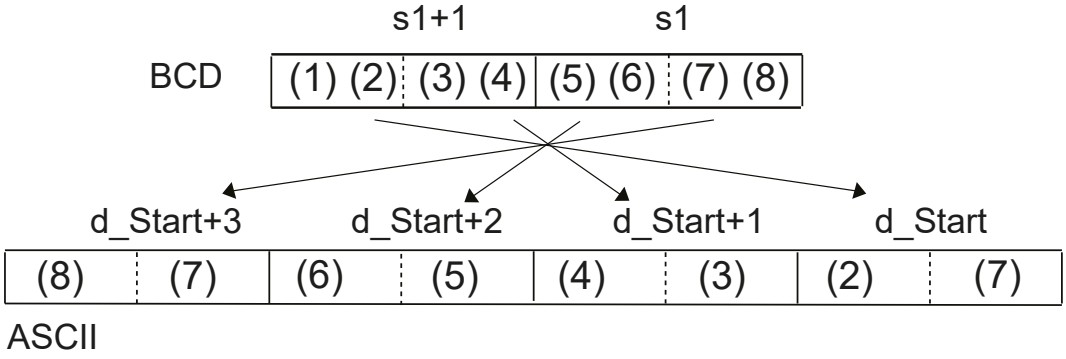
(3) conversion direction:
0: forward
1: reverse

The two characters that make up one byte are interchanged when stored. Two bytes are converted as one segment of data.

- Forward direction:




- Reverse direction:




ASCII HEX codes to express BCD characters:

| BCD character | ASCII HEX code |
|---|---|
| 0123456789 | 16#3016#31 16#32 16#33 16#34 16#35 16#36 16#37 16#38 16#39 |

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the data specified by **s1_Start** is not BCD data
- if the number of bytes specified by **s2_Control** is greater than the area specified by **s1_Start**
- if the conversion result is greater than the data area specified by **d_Start**
- if **s2_Control** = 0
- if the conversion direction is out of range
- if the number of bytes specified in **s2_Control** is greater than 4

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the data specified by **s1_Start** is not BCD data

- if the number of bytes specified by **s2_Control** is greater than the area specified by **s1_Start**
- if the conversion result is greater than the data area specified by **d_Start**
- if **s2_Control** = 0
- if the conversion direction is out of range
- if the number of bytes specified in **s2_Control** is greater than 4

### Example 16-bit data, forward direction, 2 bytes (characters)

| | Offset | BCD characters | | | Offset | Converted ASCII codes | BCD equivalent |
|---|---|---|---|---|---|---|---|
| **s1** | 0 | 16#1234 | ⇒ | **d_Start** | 0 | 16#3433 | 43 |
| **s2_Control** | | 16#0002 | | | | 16#3231 | 21 |
| | | | | | | | |

### Example 16-bit data, reverse direction, 2 bytes (characters)

| | Offset | BCD characters | | | Offset | Converted ASCII codes | BCD equivalent |
|---|---|---|---|---|---|---|---|
| **s1** | 0 | 16#1234 | ⇒ | **d_Start** | 0 | 16#3231 | 21 |
| **s2_Control** | | 16#1002 | | | | 16#3433 | 43 |
| | | | | | | | |

### Example 32-bit data, forward direction, 4 bytes (characters)

| | Offset | BCD characters | | | Offset | Converted ASCII codes | BCD equivalent |
|---|---|---|---|---|---|---|---|
| **s1** | 0 | 16#5678 | ⇒ | **d_Start** | 0 | 16#3837 | 87 |
| | 1 | 16#1234 | | | 1 | 16#3635 | 65 |
| **s2_Control** | | 16#0004 | | | | 16#3433 | 43 |
| | | | | | | 16#3231 | 21 |

### Example 32-bit data, reverse direction, 4 bytes (characters)

| | Offset | BCD characters | | | Offset | Converted ASCII codes | BCD equivalent |
|---|---|---|---|---|---|---|---|
| **s1** | 0 | 16#5678 | ⇒ | **d_Start** | 0 | 16#3231 | 21 |
| | 1 | 16#1234 | | | 1 | 16#3433 | 43 |
| **s2_Control** | | 16#1004 | | | | 16#3635 | 65 |
| | | | | | | 16#3837 | 87 |

## Example

### POU header

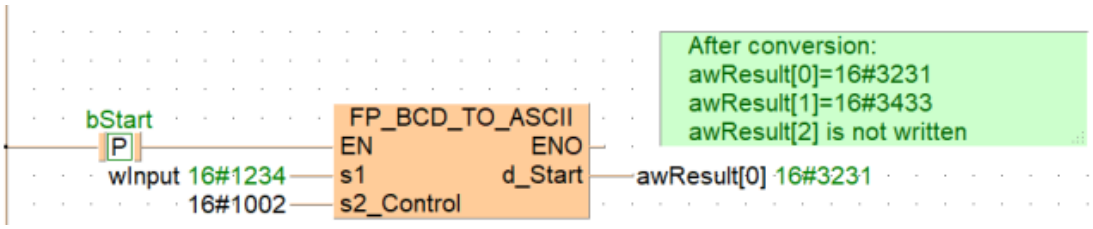
All input and output variables used for programming this function have been declared in the POU header.  The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bStart | BOOL | FALSE |
| 2 | VAR | wInput | WORD | 16#1234 |
| 3 | VAR | awResult | ARRAY [0..2] OF WORD | [3(0)] |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out. Two bytes from **s1_Start** are converted to an ASCII value in reverse direction.

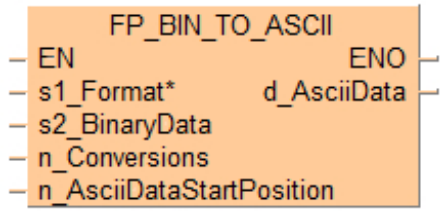
### LD body



# FP_BIN_TO_ASCII

Binary -> ASCII conversion

This FP instruction converts 16-bit/32-bit binary data stored in the area specified by **s2_BinaryData** to ASCII code. The conversion method is specified by the control string of **s1_Format**. The converted result is stored in the area specified by **d_AsciiData**.




## Parameters

Input

**s1_Format** (STRING)

Control and format string set in inverted commas

**s2_BinaryData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Starting address for storing binary data

**n_Conversions** (WORD, INT, UINT)

Quantity of numbers to be converted: 0–65535

**n_AsciiDataStartPosition** (INT)

Starting position in ASCII data: 0–255

Output

**d_AsciiData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Starting address for storing ASCII data

### Explanation of each digit of the control code s1_Format:

| Position in the control code | Description | | s1_Format | 16-bit PLCs | 32-bit PLCs |
|---|---|---|---|---|---|
| '+%08lX' | Conversion direction | | | | |
| | **+** | forward (only for hexadecimal numbers with specifier **x**, **X**) | '+%4X' | ● | ● |
| | | if plus sign is omitted: reverse (default setting) | '%4X' | ● | ● |
| '+%08lX' | **%** | format string specifier (mandatory) | | ● | ● |
| '+%08lX' | Padding format | | | | |
| | **0** | fill with zeros | '%06x' | - | ● |
| | **+** | add a plus sign | '%+4d' | - | ● |
| | **-** | left alignment | '%-6d' | - | ● |
| | **␣** | (space) add a space instead of plus sign | '%␣4d' | - | ● |
| | # | insert 0x for hexadecimal numbers | '%#4X' | - | ● |
| | | append always a decimal point for real number | '%#8.0f' | - | ● |
| '+%08lX' | **8** | width of the ASCII data element | '%08d' | ● | ● |
| | no width<br>• **FP_BIN_TO_ASCII**: required minimum width is assumed<br>• **FP_ASCII_TO_BIN** and **FP_ASCII_CHECK**: comma separator is required | | '%d,' | - | ● |
| '+%#8.5lx' | Precision after decimal point | | | | |
| | **.5** | any digit after decimal point | '%8.5f' | - | ● |
| '+%08lX' | **l** | double length specifier<br>e.g. specifier i with l = li -> DINT | '+%4ld' | ● | ● |
| '+%08lX' | Format specifier | | | | |
| | **i** | INT | '%10i' | ● | ● |
| | **u** | UINT | '%10u' | - | ● |
| | **d** | INT | '%6d' | ● | ● |
| | **x** | hexadecimal lower case | '+%4x' | ● | ● |
| | **X** | hexadecimal upper case | '+%4X' | ● | ● |
| | **b** | BCD | '+%5b' | - | ● |
| | **f** | floating-point number | '+%-6.2f' | - | ● |
| | **e** | exponential 1.23e10 | '+%9.3e' | - | ● |
| | **E** | exponential upper case 1.23E10 | '+%9.3E' | - | ● |

| Position in the control code | Description | | s1_Format | 16-bit PLCs | 32-bit PLCs |
|---|---|---|---|---|---|
| | **g** | floating or exponential | '+%12g' | - | ● |
| | **G** | floating or exponential upper case | '+%9.3G' | - | ● |
| '%8dPANA' | • For **FP_BIN_TO_ASCII**: any string can be appended to the conversion e.g. 'PANA'<br>• For **FP_ASCII_TO_BIN** and **FP_ASCII_CHECK**: only comma is allowed | | '+%8dPANA' | - | ● |

● available on 16-bit/32-bit PLCs

### Examples of control string s1_Format

- Example: '+%4X' converts a binary value to a value with a width of four characters in upper case in forward direction (valid for 16-bit/32-bit PLCs)

| Binary data | Conversion result in ASCII data |
|---|---|
| 16#12A | '␣12A' |

- Example: '%4X' converts a binary value to a value with a width of four characters in upper case in reverse direction (valid for 16-bit/32-bit PLCs)

| Binary data | Conversion result in ASCII data |
|---|---|
| 16#B2A | '␣B2A' |

- Example: '%06d' converts a binary value to a value with a width of six characters, decimal value with three leading zeros (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| 16#100 | '000100' |

- Example: '%+4d' converts a binary value to a value with a width of four characters, decimal value, + sign added (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| 100 | '+100␣␣' |

- Example: '%-6d' converts a binary value to a value with a width of six characters, left aligned (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| 100 | '100␣␣␣' |

- Example: `'%␣4d'` converts a binary value to a value with a width of four characters with one leading space (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| 100 | '␣100' |

- Example: `'%#8.0f'` converts a binary value to a value with a width of eight characters, floating-point number, four leading spaces (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| 123.45678 | '␣␣␣␣123.' |

- Example: `'%8.3f'` converts a binary value to a value with a width of eight characters, three characters precision after decimal point (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| 123.45599 | '␣123.456' |

- Example: `'+%10u'` converts a binary value to a value with a width of 10 characters, seven leading spaces, unsigned integer (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| -100 | '␣␣␣␣␣␣␣100' |

- Example: `'%06d'` converts a binary value to a value with a width of six characters with three leading spaces (valid for 16-bit/32-bit PLCs)

| Binary data | Conversion result in ASCII data |
|---|---|
| 100 | '␣␣␣100' |

Permissible range for the value before the specifier:

1–15 before specifiers **d**, **ld**, **i**, **li**

1–4 before specifier **X**

1–8 before specifier **lX**

if no width is specified, comma is appended e.g. `'%d,'` (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| 100 | '100,' |

  - **FP_BIN_TO_ASCII**: required minimum width is assumed
  - **FP_ASCII_TO_BIN** and **FP_ASCII_CHECK**: comma separator is required

- Example: `'+%4ld'` converts a binary value to a value with a width of four characters, requires DINT or DWORD for the converted result (valid for 16-bit/32-bit PLCs)

| Binary data | Conversion result in ASCII data |
|---|---|
| 100 | '␣100' |

- Example: `'+%6i'` converts a binary value to a value with a width of six characters, three leading spaces, signed integer in forward direction (valid for 16-bit/32-bit PLCs)

| Binary data | Conversion result in ASCII data |
|---|---|
| -100 | '␣␣␣-100' |

- Example: `'+%6d'` converts a binary value to a value with a width of six characters with three leading zeros in forward direction (valid for 16-bit/32-bit PLCs)

| Binary data | Conversion result in ASCII data |
|---|---|
| 100 | '000100' |

- Example: `'+%4X'` converts a binary value to a value with a width of four characters, hexadecimal number in upper case in forward direction (valid for 16-bit/32-bit PLCs)

| Binary data | Conversion result in ASCII data |
|---|---|
| 16#12A | '␣12A' |

- Example: `'+%5b'` converts a binary value to a value with a width of five characters, BCD data (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| 16#123 | '␣123' |

- Example: `'+%-6.2f'` converts a binary value to a value with a width of six characters, left aligned, precision two digits after decimal point (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| 1.2345 | '1.23␣' |

- Example: `'+%9.3e'` converts a binary value to a value with a width of 9 characters, precision 3 digits after decimal point, exponential lower case (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| 1234.5678 | '1.235e+03' |

- Example: `'1.235E+03'` converts a binary value to a value with a width of 9 characters, precision 3 digits after decimal point, exponential upper case (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| 1234.5678 | '1.235E+03' |

- Example: `'+%12g'` converts a binary value to a value with a width of 12 characters, floating-point number (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| 1234.5678 | '␣␣␣␣␣1234.57' |

- Example: `'+%9.3G'` converts a binary value to a value with a width of nine characters, precision three characters after decimal point, floating-point number, exponential upper case (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| 1234 | '␣␣␣1.E+03' |

- For **FP_BIN_TO_ASCII**: any string can be appended to the conversion
  Example: `'+%8dPANA'` converts a binary value to a value with a width of eight characters, decimal number, 'PANA' is appended (valid for 32-bit PLCs only)

| Binary data | Conversion result in ASCII data |
|---|---|
| 100 | '␣␣␣␣100PANA' |

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if there is an error in the control string specified by **sFormat**.

- if forward direction (+) is specified in **sFormat** when the format is decimal.
- if the number of ASCII characters per converted unit specified by **n_Conversions** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specified by **s1_Format**.
- if 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_Conversions**.
- if the number of 16- or 32-bit decimal numbers to be converted specified by **n_Conversions** exceeds the area for storing ASCII data.
- if the converted result exceeds the area.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if there is an error in the control string specified by **sFormat**.
- if forward direction (+) is specified in **sFormat** when the format is decimal.
- if the number of ASCII characters per converted unit specified by **n_Conversions** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specified by **s1_Format**.
- if 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_Conversions**.
- if the number of 16- or 32-bit decimal numbers to be converted specified by **n_Conversions** exceeds the area for storing ASCII data.
- if the converted result exceeds the area.

## Example

### POU header

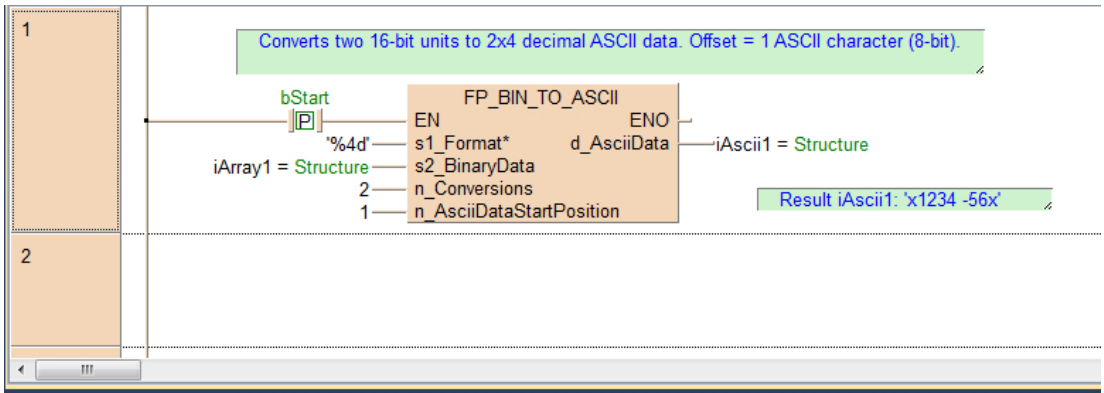
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | iArray1 | ARRAY [0..1] OF INT | [1234,-56] |
| 2 | VAR | iAscii1 | ARRAY [0..4] OF WORD | [5(16#FFFF)] |

### POU body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out. It converts two 16-bit units to 2 x 4 decimal ASCII data. Offset = 1 ASCII character (8-bit).

## LD body



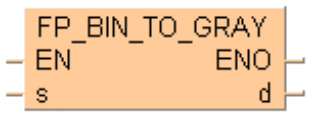

FP_BIN_TO_ASCII_LD

| | |
|---|---|
| -FP_BIN_TO_ASCII_LD | Structure |
| bStart | 2#1 at R1020 |
| -iArray1 | Structure |
| [0] | 1234 at DT9830 |
| [1] | -56 at DT9831 |
| -iAscii1 | Structure |
| [0] | 'ÿ1' at DT9832 |
| [1] | '23' at DT9833 |
| [2] | '4 ' at DT9834 |
| [3] | '-5' at DT9835 |
| [4] | '6ÿ' at DT9836 |

# FP_BIN_TO_GRAY

Binary -> gray code conversion

This FP instruction converts the binary data at input **s** into a gray code value if the trigger **EN** is TRUE. The result is stored in **d**.




## Parameters

Input

**s** (WORD, DWORD)

Binary data

Output

**d** (WORD, DWORD)

Gray code value

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## POU header

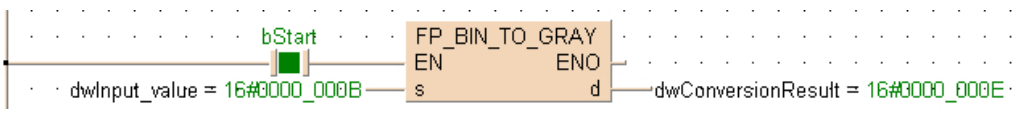
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the fuction |
| 1 | VAR | dwInput_value | DWORD | 16#0000000B | |
| 2 | VAR | dwConversionResult | DWORD | 0 | result after a 0->1 leading |

## POU body

When the variable **bStart** is set to TRUE, the function is carried out.

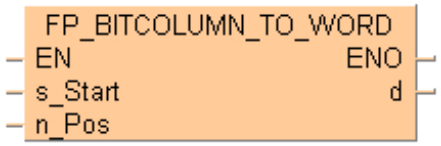
## LD body



# FP_BITCOLUMN_TO_WORD

Bit column -> bit line conversion

This FP instruction converts a bit column out of an address range starting at input **s_Start** and returns each bit in a bit line at output **d**. The bit position is specified at input **n_Pos** (range 0–15). This instruction is the reversion of **FP_WORD_TO_BITCOLUMN**.




## Parameters

Input

**s_Start** (WORD)

Starting address of the bit column

**n_Pos** (WORD, INT, UINT)

Bit position (range 0–15)

Output

**d** (WORD)

Result

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if 0 ≥ **n_Pos** > 15
- if **s_Start** is out of range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if 0 ≥ **n_Pos** > 15
- if **s_Start** is out of range

## Example

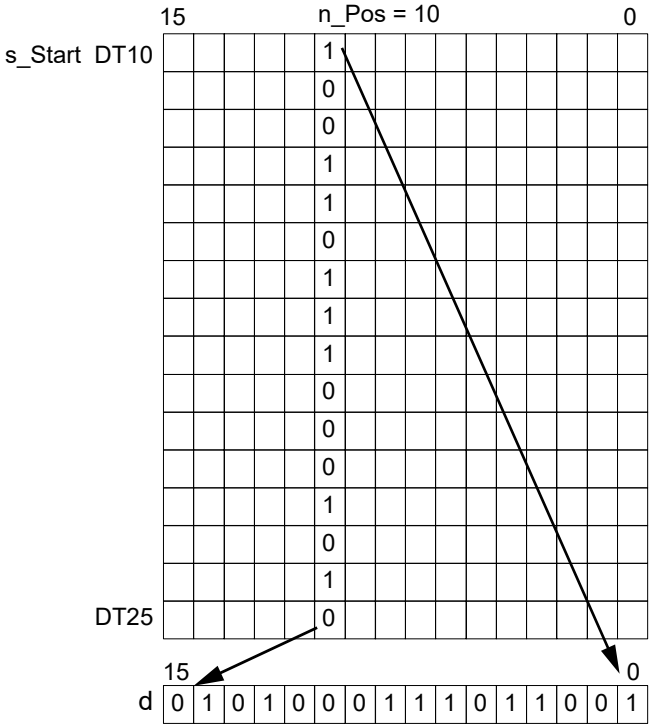
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

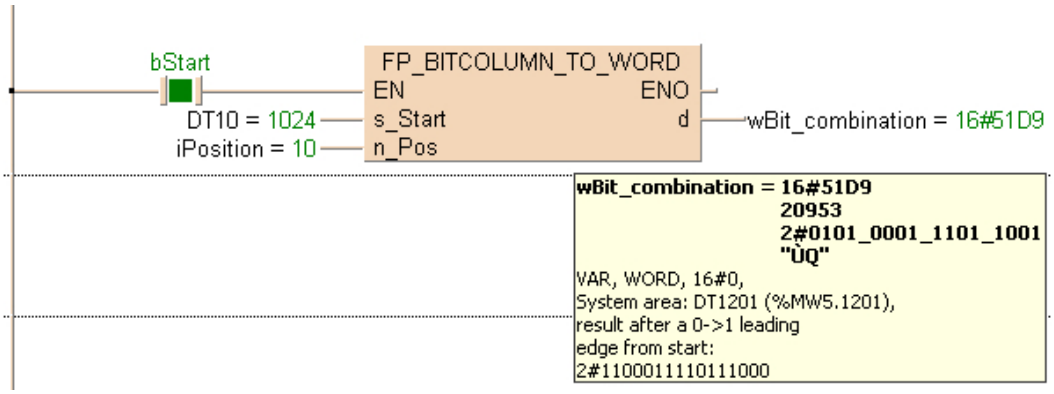
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | iPosition | INT | 10 | specifies the position |
| 2 | VAR | wBit_combination | WORD | 16#0 | result after a 0->1 leading |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out. The bit column at bit position 10 of the data starting at DT10 is output as 0101 0001 1101 1001 at **d**.



### LD body

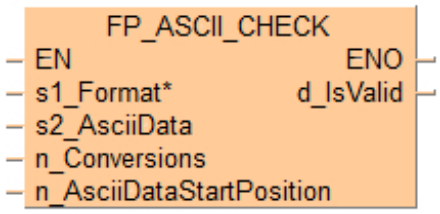

bStart
FP_BITCOLUMN_TO_WORD
EN
ENO
DT10 = 1024
s_Start
d
wBit_combination = 16#51D9
iPosition = 10
n_Pos
wBit_combination = 16#51D9
20953
2#0101_0001_1101_1001
"ÙQ"
VAR, WORD, 16#0,
System area: DT1201 (%MW5.1201),
result after a 0->1 leading
edge from start:
2#1100011110111000

# FP_ASCII_CHECK

ASCII data check

This FP instruction checks whether the ASCII codes stored in the area specified by **s2_AsciiData** can be converted correctly using the conversion method specified by **s1_Format**.

- If the results are correct, the system variable **sys_bIsEqual** and the output **d_IsValid** turns on.
- If the results are incorrect, the system variable **sys_bIsEqual** and the output **d_IsValid** turns off.




Note

The FP instructions **FP_ASCII_TO_BIN** and **FP_ASCII_CHECK** do not differentiate between upper and lower case.

## Parameters

Input

**s1_Format** (STRING)

Control and format string set in inverted commas

**s2_AsciiData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Starting address for storing ASCII data

**n_Conversions** (WORD, INT, UINT)

Quantity of numbers to be converted: 0–65535

**n_AsciiDataStartPosition** (INT)

Starting position in ASCII data: 0–255

Output

**d_IsValid** (BOOL)

TRUE, if ASCII data consits of binary numbers that can be converted into the desired format

### Explanation of each digit of the control code s1_Format:

| Position in the control code | Description | | s1_Format | 16-bit PLCs | 32-bit PLCs |
|---|---|---|---|---|---|
| '+%08lX' | Conversion direction | | | | |
| | **+** | forward (only for hexadecimal numbers with specifier **x**, **X**) | '+%4X' | ● | ● |
| | | if plus sign is omitted: reverse (default setting) | '%4X' | ● | ● |
| '+%08lX' | **%** | format string specifier (mandatory) | | ● | ● |
| '+%08lX' | Padding format | | | | |
| | **0** | fill with zeros | '%06x' | - | ● |
| | **+** | add a plus sign | '%+4d' | - | ● |
| | **-** | left alignment | '%-6d' | - | ● |
| | **␣** | (space) add a space instead of plus sign | '%␣4d' | - | ● |
| | # | insert 0x for hexadecimal numbers | '%#4X' | - | ● |
| | | append always a decimal point for real number | '%#8.0f' | - | ● |
| '+%08lX' | **8** | width of the ASCII data element | '%08d' | ● | ● |
| | no width<br>• **FP_BIN_TO_ASCII**: required minimum width is assumed<br>• **FP_ASCII_TO_BIN** and **FP_ASCII_CHECK**: comma separator is required | | '%d,' | - | ● |
| '+%#8.5lx' | Precision after decimal point | | | | |
| | **.5** | any digit after decimal point | '%8.5f' | - | ● |
| '+%08lX' | **l** | double length specifier<br>e.g. specifier i with l = li -> DINT | '+%4ld' | ● | ● |
| '+%08lX' | Format specifier | | | | |
| | **i** | INT | '%10i' | ● | ● |
| | **u** | UINT | '%10u' | - | ● |
| | **d** | INT | '%6d' | ● | ● |
| | **x** | hexadecimal lower case | '+%4x' | ● | ● |
| | **X** | hexadecimal upper case | '+%4X' | ● | ● |
| | **b** | BCD | '+%5b' | - | ● |
| | **f** | floating-point number | '+%-6.2f' | - | ● |
| | **e** | exponential 1.23e10 | '+%9.3e' | - | ● |
| | **E** | exponential upper case 1.23E10 | '+%9.3E' | - | ● |

| Position in the control code | Description | | s1_Format | 16-bit PLCs | 32-bit PLCs |
|---|---|---|---|---|---|
| | **g** | floating or exponential | '+%12g' | - | ● |
| | **G** | floating or exponential upper case | '+%9.3G' | - | ● |
| '%8dPANA' | • For **FP_BIN_TO_ASCII**: any string can be appended to the conversion e.g. 'PANA'<br>• For **FP_ASCII_TO_BIN** and **FP_ASCII_CHECK**: only comma is allowed | | '+%8dPANA' | - | ● |

● available on 16-bit/32-bit PLCs

### Examples of control string s1_Format

- Example: '+%4X' converts a binary value to a value with a width of four characters in upper case in forward direction (valid for 16-bit/32-bit PLCs)
  Valid ASCII data for this example: '␣12A'

- Example: '%4X' converts a binary value to a value with a width of four characters in upper case in reverse direction (valid for 16-bit/32-bit PLCs)
  Valid ASCII data for this example: '␣B2A'

- Example: '%06d' converts a binary value to a value with a width of six characters, decimal value with three leading zeros (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '000100'

- Example: '%+4d' converts a binary value to a value with a width of four characters, decimal value, + sign added (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '+100␣␣'

- Example: '%-6d' converts a binary value to a value with a width of six characters, left aligned (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '100␣␣␣'

- Example: '%␣4d' converts a binary value to a value with a width of four characters with one leading space (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '␣100'

- Example: '%#8.0f' converts a binary value to a value with a width of eight characters, floating-point number, four leading spaces (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '␣␣␣␣123.'

- Example: `'%8.3f'` converts a binary value to a value with a width of eight characters, three characters precision after decimal point (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '␣123.456'

- Example: `'+%10u'` converts a binary value to a value with a width of 10 characters, seven leading spaces, unsigned integer (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '␣␣␣␣␣␣␣100'

- Example: `'%06d'` converts a binary value to a value with a width of six characters with three leading spaces (valid for 16-bit/32-bit PLCs)
  Valid ASCII data for this example: '␣␣␣100'
  Permissible range for the value before the specifier:
  '␣␣␣␣␣100PANA'
  if no width is specified, comma is appended e.g. `'%d,'` (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '100,'
  - **FP_BIN_TO_ASCII**: required minimum width is assumed
  - **FP_ASCII_TO_BIN** and **FP_ASCII_CHECK**: comma separator is required

- Example: `'+%4ld'` converts a binary value to a value with a width of four characters, requires DINT or DWORD for the converted result (valid for 16-bit/32-bit PLCs)
  Valid ASCII data for this example: '␣100'

- Example: `'+%6i'` converts a binary value to a value with a width of six characters, three leading spaces, signed integer in forward direction (valid for 16-bit/32-bit PLCs)
  Valid ASCII data for this example: '␣␣␣-100'

- Example: `'+%6d'` converts a binary value to a value with a width of six characters with three leading zeros in forward direction (valid for 16-bit/32-bit PLCs)
  Valid ASCII data for this example: '000100'

- Example: `'+%4X'` converts a binary value to a value with a width of four characters, hexadecimal number in upper case in forward direction (valid for 16-bit/32-bit PLCs)
  Valid ASCII data for this example: '␣12A'

- Example: `'+%5b'` converts a binary value to a value with a width of five characters, BCD data (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '␣123'

- Example: `'+%-6.2f'` converts a binary value to a value with a width of six characters, left aligned, precision two digits after decimal point (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '1.23␣'

- Example: `'+%9.3e'` converts a binary value to a value with a width of 9 characters, precision 3 digits after decimal point, exponential lower case (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '1.235e+03'

- Example: `'1.235E+03'` converts a binary value to a value with a width of 9 characters, precision 3 digits after decimal point, exponential upper case (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '1.235E+03'

- Example: `'+%12g'` converts a binary value to a value with a width of 12 characters, floating-point number (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '␣␣␣␣␣1234.57'

- Example: `'+%9.3G'` converts a binary value to a value with a width of nine characters, precision three characters after decimal point, floating-point number, exponential upper case (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '␣␣␣1.E+03'

- For **FP_BIN_TO_ASCII**: any string can be appended to the conversion
  Example: `'+%8dPANA'` converts a binary value to a value with a width of eight characters, decimal number, 'PANA' is appended (valid for 32-bit PLCs only)
  Valid ASCII data for this example: '␣␣␣␣␣100PANA'

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if there is an error in the control string specified by **sFormat**.
- if forward direction (+) is specified in **sFormat** when the format is decimal.
- if the number of ASCII characters per converted unit specified by **n_Conversions** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specified by **s1_Format**.
- if 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_Conversions**.
- if the number of 16- or 32-bit decimal numbers to be converted specified by **n_Conversions** exceeds the area for storing ASCII data.
- if the converted result exceeds the area.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if there is an error in the control string specified by **sFormat**.
- if forward direction (+) is specified in **sFormat** when the format is decimal.
- if the number of ASCII characters per converted unit specified by **n_Conversions** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specified by **s1_Format**.

- if 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_Conversions**.
- if the number of 16- or 32-bit decimal numbers to be converted specified by **n_Conversions** exceeds the area for storing ASCII data.
- if the converted result exceeds the area.

## Example

### POU header

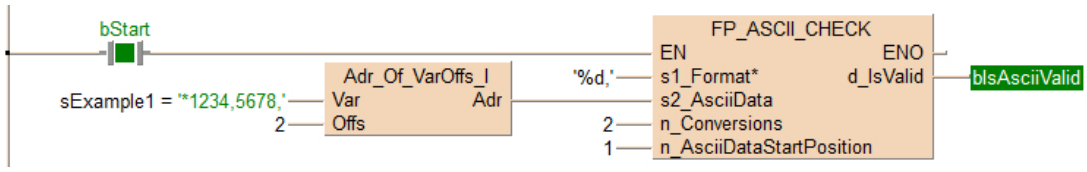
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | sExample1 | STRING[32] | '*1234,5678,' |
| 2 | VAR | bIsAsciiValid | BOOL | FALSE |

### POU body

When **bStart** is set to TRUE, the instruction checks whether the data connected at **s2_AsciiData** can be converted to decimal data when the format string is '%d'.

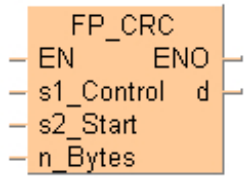
### LD body



# FP_CRC

Cyclic redundancy check

This FP instruction calculates the cyclic redundancy check (CRC) according to the calculation specified by **s1_Control**. The data is specified by the start address **s2_Start** and the number of bytes **n_Bytes**. The CRC value is stored in the 16-bit area specified by **d**.




## Parameters

Input

**s1_Control** (WORD, INT, UINT)

Calculation method

**s2_Start** (WORD, INT, UINT)

Starting address

**n_Bytes** (WORD, INT, UINT)

Number of bytes

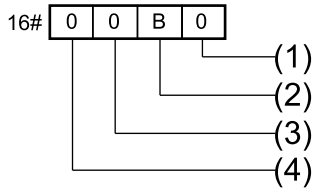
Output

**d** (WORD, INT, UINT)

CRC value

## Remarks

Specifying the control code **s1_Control**




(1) Calculation method
0: CRC (**SYS_CRC_CALCULATION_METHOD_CRC16**)
1: CRC-CCITT (**SYS_CRC_CALCULATION_METHOD_CRC16_CCITT**)

(2) Calculation starting position (bytes from **s2_Start**)
16#0–16#F

(3) Storage starting position (bytes from **d**)
16#0–16#F

(4) Data to be converted
16#0: Binary data (1 byte)

Generator polynomials

CRC-16: $x^{16} + x^{15} + x^2 + 1$

CRC-CCITT: $x^{16} + x^{12} + x^5 + 1$

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the calculation method or conversion data specified by **s1_Control** is out of range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the calculation method or conversion data specified by **s1_Control** is out of range

## Example

## POU header

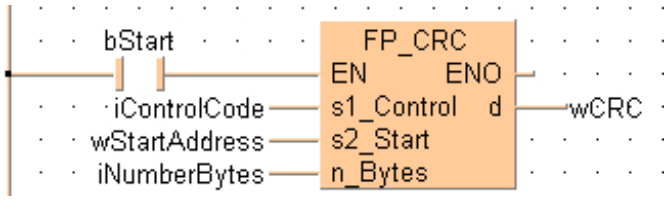
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | wStartAddress | WORD | 16#1002 | |
| 2 | VAR | iNumberBytes | INT | 22 | |
| 3 | VAR | wCRC | WORD | 0 | |
| 4 | VAR | iControlCode | INT | 16#00B0 | |

## POU body

When the variable **bStart** is set to TRUE, the function is carried out.

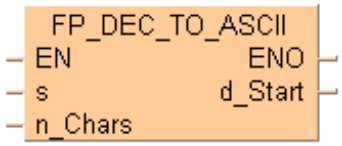
### LD body


bStart
FP_CRC
EN
ENO
iControlCode
s1_Control
d
wCRC
wStartAddress
s2_Start
iNumberBytes
n_Bytes

# FP_DEC_TO_ASCII

Decimal -> ASCII conversion

This FP instruction converts the decimal characters specified by **s** to ASCII codes if the trigger **EN** is TRUE. **n_Chars** specifies the number of bytes to be converted.The result is stored in the area specified by **d_Start**.




ASCII HEX codes to express decimal characters:

| Decimal characters | ASCII HEX code |
|---|---|
| SPACE+-0123456789 | 16#2016#2B16#2D16#3016#3116#3216#3316#3416#3516#3616#3716#3816#39 |

## Parameters

Input

**s** (INT, DINT, UINT, UDINT)

Input value

**n_Chars** (WORD, INT, UINT)

Number of bytes

Output

**d_Start** (WORD, INT, UINT)

Starting address of the data area for the results. The size is **n_Bits** * 2 words.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the number of bytes specified by **n_Chars** is greater than the area specified by **d_Start**.
- if the conversion result is greater than the data area specified by **d_Start**
- if the number of bytes of the conversion result is greater than the number of bytes specified by **n_Chars**.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the number of bytes specified by **n_Chars** is greater than the area specified by **d_Start**.
- if the conversion result is greater than the data area specified by **d_Start**
- if the number of bytes of the conversion result is greater than the number of bytes specified by **n_Chars**.

### Example 16-bit data, unsigned

| | Offset | Dec. characters | Decimal figure | | | Offset | Converted ASCII codes | Dec. equivalent |
|---|---|---|---|---|---|---|---|---|
| **s** | 0 | 16#FF9C | 65436 | ⇒ | **d_Start** | 0 | 16#3620 | 6 |
| **n_Chars** | 0 | 16#0006 | | | | 1 | 16#3435 | 45 |
| | | | | | | 2 | 16#3633 | 63 |

### Example 16-bit data, signed

| | Offset | Dec. characters | Decimal figure | | | Offset | Converted ASCII codes | Dec. equivalent |
|---|---|---|---|---|---|---|---|---|
| **s** | 0 | 16#FF9C | -100 | ⇒ | **d_Start** | 0 | 16#2020 | |
| **n_Chars** | 0 | 16#0006 | | | | 1 | 16#312D | 1 - |
| | | | | | | 2 | 16#3030 | 00 |

### Example 32-bit data, unsigned

| | Offset | Dec. characters | Decimal figure | | | Offset | Converted ASCII codes | Dec. equivalent |
|---|---|---|---|---|---|---|---|---|
| **s** | 0 | 16#FF9C | 4294967196 | ⇒ | **d_Start** | 0 | 16#3234 | 24 |
| | 1 | 16#FFFF | | | | 1 | 16#3439 | 49 |
| **n_Chars** | 0 | 16#000A | | | | 2 | 16#3639 | 69 |
| | | | | | | 3 | 16#3137 | 17 |
| | | | | | | 4 | 16#3639 | 69 |

### Example 32-bit data, signed

| | Offset | Dec. characters | Decimal figure | | | Offset | Converted ASCII codes | Dec. equivalent |
|---|---|---|---|---|---|---|---|---|
| **s** | 0 | 16#FF9C | -100 | ⇒ | **d_Start** | 0 | 16#2020 | 0 |
| | 1 | 16#FFFF | | | | 1 | 16#2020 | 0 |
| **n_Chars** | 0 | 16#000A | | | | 2 | 16#2020 | 0 |
| | | | | | | 3 | 16#312D | 1 - |
| | | | | | | 4 | 16#3030 | 00 |

## Example

### POU header

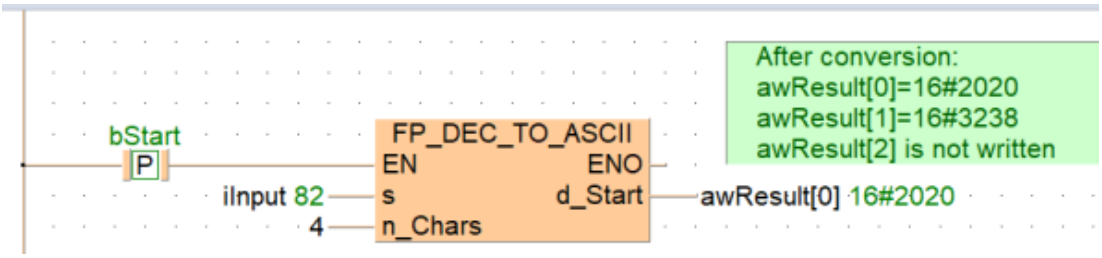
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 1 | VAR | bStart | BOOL | FALSE | |
| 2 | VAR | iInput | INT | 82 | |
| 3 | VAR | awResult | ARRAY [0..2] OF WORD | [3(0)] | |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

### LD body

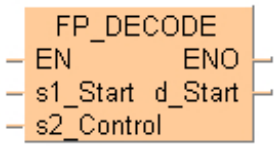


# FP_DECODE

Decode hexadecimal -> bit state

This FP instruction decodes the contents of data specified by **s1_Start** according to the contents of **s2_Control** if the trigger **EN** is set to TRUE. The decoded result is stored in the area starting from **d_Start**.

**FP_DECODE** is the inverse instruction of **FP_ENCODE**.




## Parameters

Input

**s1_Start** (WORD)

Source area or equivalent constant to be decoded

**s2_Control** (WORD)

Control data to specify the starting bit position and number of bits to be decoded

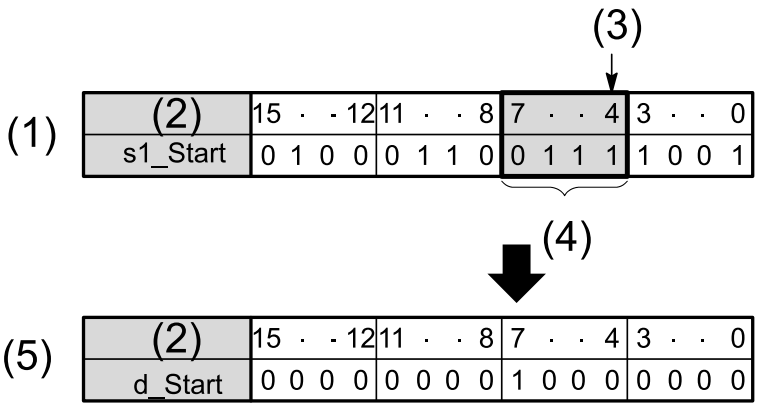
Output

**d_Start** (WORD)

Starting address for storing decoded data (destination)

## Remarks

- The variables **s1_Start**, **s2_Control** and **d_Start** have to be of the same data type.
- Example**s2_Control**=16#404




(1) Source

(2) Bit position

(3) Bit position 4

(4) 4 bits (bit 4–7, binary: 2#0111 or decimal: 7) of **s1_Start** are decoded when the trigger is TRUE

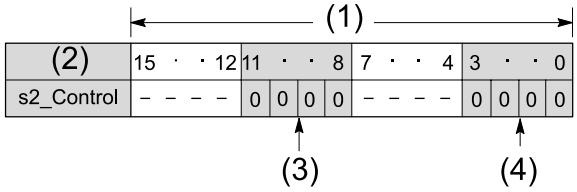
(5) Destination
The decoded data TRUE is stored in bit position 7 of **d_Start**.
All bits except bit 7 are filled with zeros.

The contents of the 2^nL bit segment from nH bit of the area specified by **s1_Start** are decoded (where nL is the number of bits to be decoded).

The decoded results are stored in the 2^nL bit segments starting from the area specified by **d_Start** (where nH is the conversion start bit). Invalid bits in the specified area for the result are filled with zeros.

- Description of **s2_Control**

  **s2_Control** specifies the starting bit position and the number of bits to be decoded using hexadecimal data.




(1) 16-bit data

(2) Number of bit 0–15

The bits marked "-" are invalid.

| Set value 16#0–16#F | (3) Starting bit position to be decoded | | Set value 16#0–16#8 | (4) Number of decoded bits |
|---|---|---|---|---|
| 16#0 | 0 | | 16#0 | 0 |
| 16#1 | 1 | | 16#1 | 1 |
| 16#2 | 2 | | 16#2 | 2 |
| 16#3 | 3 | | 16#3 | 3 |
| 16#4 | 4 | | 16#4 | 4 |
| 16#5 | 5 | | 16#5 | 5 |
| 16#6 | 6 | | 16#6 | 6 |
| 16#7 | 7 | | 16#7 | 7 |
| 16#8 | 8 | | 16#8 | 8 |
| 16#9 | 9 | | | |
| 16#A | 10 | | | |
| 16#B | 11 | | | |
| 16#C | 12 | | | |
| 16#D | 13 | | | |
| 16#E | 14 | | | |
| 16#F | 15 | | | |

- Relationship between number of bits and occupied data area for decoded result

| **Number of bits to be decoded** | **Data area required for the result** | **Number of decoded bits (result)** |
|---|---|---|
| 1 | 1 word | 2-bit [1)] |
| 2 | 1 word | 4-bit [1)] |
| 3 | 1 word | 8-bit [1)] |
| 4 | 1 word | 16-bit |

| Number of bits to be decoded | Data area required for the result | Number of decoded bits (result) |
|---|---|---|
| 5 | 2 words | 32-bit |
| 6 | 4 words | 64-bit |
| 7 | 8 words | 128-bit |
| 8 | 16 words | 256-bit |

[1)] Invalid bits in the data area required for the result are filled with zeros.

- Example of decoded data

  When decoding 4-bit data, 16-bit data for the decoded result is shown below.

  – Decoding condition: **s2_Control**

  – Starting bit position 0: 16#0

  – Number of bits to be decoded: 16#4 (4 bits)

| Data to be decoded (4 bits) | | Decoded result |
|---|---|---|
| Hex. | Dec. | |
| 16#0000 | 0 | 0000 0000 0000 0001 |
| 16#0001 | 1 | 0000 0000 0000 0010 |
| 16#0010 | 2 | 0000 0000 0000 0100 |
| 16#0011 | 3 | 0000 0000 0000 1000 |
| 16#0100 | 4 | 0000 0000 0001 0000 |
| 16#0101 | 5 | 0000 0000 0010 0000 |
| 16#0110 | 6 | 0000 0000 0100 0000 |
| 16#0111 | 7 | 0000 0000 1000 0000 |
| 16#1000 | 8 | 0000 0001 0000 0000 |
| 16#1001 | 9 | 0000 0010 0000 0000 |
| 16#1010 | 10 | 0000 0100 0000 0000 |
| 16#1011 | 11 | 0000 1000 0000 0000 |
| 16#1100 | 12 | 0001 0000 0000 0000 |
| 16#1101 | 13 | 0010 0000 0000 0000 |
| 16#1110 | 14 | 0100 0000 0000 0000 |
| 16#1111 | 15 | 1000 0000 0000 0000 |

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if $1 \leq nL \leq 8$, where nL is the number of bits to be encoded/decoded.

- if 1 ≤ nH + nL ≤ 16, where nH is the conversion start bit and nL is the number of bits to be encoded/decoded.
- if the memory area used by the decoded result exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if 1 ≤ nL ≤ 8, where nL is the number of bits to be encoded/decoded.
- if 1 ≤ nH + nL ≤ 16, where nH is the conversion start bit and nL is the number of bits to be encoded/decoded.
- if the memory area used by the decoded result exceeds the limit.

## Example

### POU header

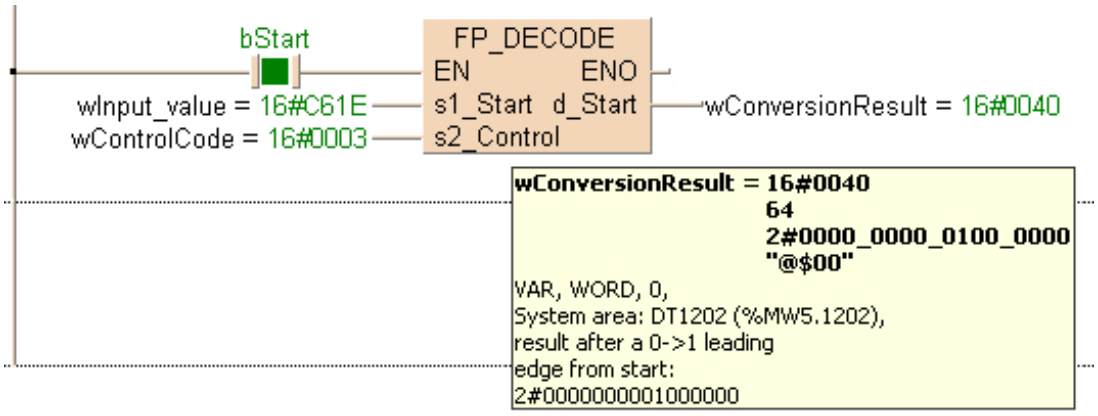
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | wInput_value | WORD | 2#1100011000011110 | |
| 2 | VAR | wControlCode | WORD | 16#0003 | specifies decoding |
| 3 | VAR | wConversionResult | WORD | 0 | result after a 0->1 leading |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

### LD body



# FP_DEG

Angle units (radians -> degrees) conversion

This FP instruction converts the value of an angle entered at input **s** from radians to degrees and returns the result at output **d**.


## Parameters

Input

**s** (ANY_REAL)

Angle value

Output

**d** (ANY_REAL)

Converted angle value

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s** is not a REAL number

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s** is not a REAL number

## Example

## POU header

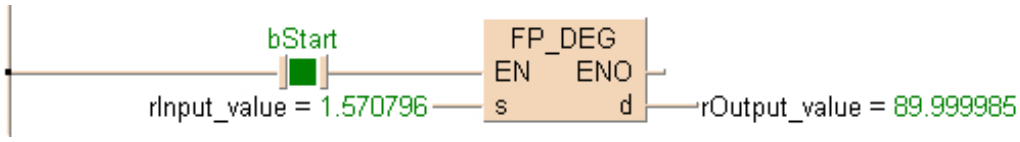
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | rInput_value | REAL | 1.570796 | corresponds to PI/2 |
| 2 | VAR | rOutput_value | REAL | 0.0 | result after a 0->1 leading |

## POU body

When the variable **bStart** is set to TRUE, the function is carried out.

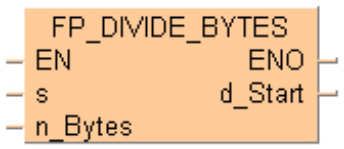
## LD body



# FP_DIVIDE_BYTES

Byte distribution

This FP instruction disintegrates the input value at **s** into the number of bytes specified at **n_Bytes** and distributes the bytes to the addresses starting at **d_Start**.




## Parameters

Input

**s** (BOOL, WORD, DWORD)

Input value

**n_Bytes** (INT, DINT, UINT, UDINT)

- Number of bytes to be distributed, range: 0–65535
- For **n_Bytes**=0, the instruction is not executed

Output

**d_Start** (WORD)

Starting address of the data area for the results. The size is **n_Bits** * 2 words.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

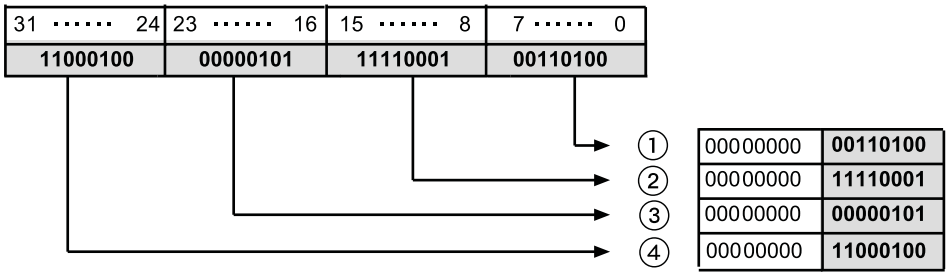
- if the area specified using the index modifier exceeds the limit.
- if the value at **n_Bytes** is out of range
- if the value at **n_Bytes** yields a result which is greater than the data area specified by **d_Start**

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the value at **n_Bytes** is out of range
- if the value at **n_Bytes** yields a result which is greater than the data area specified by **d_Start**

### Example

The bit data equivalent to 16#C405F134 is disintegrated into 4 bytes.




(1) Offset 0
(2) Offset 1
(3) Offset 2
(4) Offset 3

## Example

### POU header

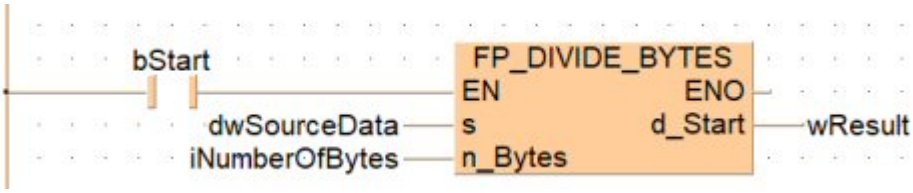
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bStart | BOOL | FALSE |
| 2 | VAR | dwSourceData | DWORD | 16#C405F134 |
| 3 | VAR | iNumberOfBytes | INT | 4 |
| 4 | VAR | wResult | WORD | 0 |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

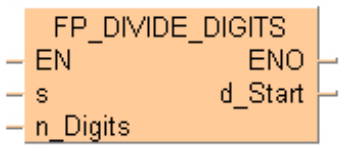
### LD body



# FP_DIVIDE_DIGITS

Digit distribution

This FP instruction disintegrates the input value at **s** into the number of digits specified at **n_Digits** and distributes the digits to the addresses starting at **d_Start**.




## Parameters

Input

**s**(WORD)

Input value

**n_Digits** (WORD, INT, UINT)

- Number of digits to be distributed, range: 0–4
- For **n_Bytes**=0, the instruction is not executed

Output

**d_Start**(WORD)

Starting address of the data area for the results. The size is **n_Bits** * 2 words.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the value at **n_Bytes** is out of range
- if the value at **n_Bytes** yields a result which is greater than the data area specified by **d_Start**

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the value at **n_Bytes** is out of range
- if the value at **n_Bytes** yields a result which is greater than the data area specified by **d_Start**

## Example

### POU header

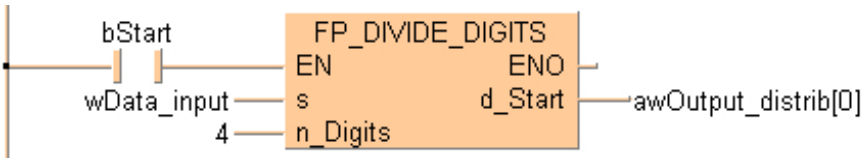
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | TRUE |
| 1 | VAR | wData_input | WORD | 16#7310 |
| 2 | VAR | awOutput_distrib | ARRAY [0..3] OF WORD | [4(0)] |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

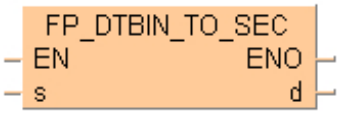
### LD body



# FP_DTBIN_TO_SEC

Binary date and time -> seconds

**FP_DTBIN_TO_SEC** converts a binary value of the data type DTBIN into seconds.




## Parameters

Input

**s** (DTBIN)

date and time in binary format

Output

**d** (DWORD, DINT, UDINT, DATE, TOD, DT)

Seconds

## Example

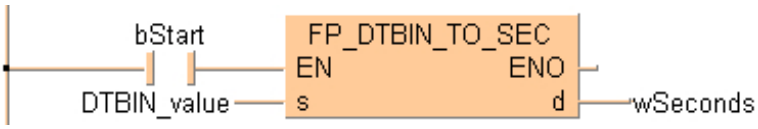
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | DTBIN_value | DTBIN | Year_2digits := 14, |
| 2 | VAR | wSeconds | WORD | 0 |

## LD body

When the variable **bStart** is set to TRUE, the function is carried out.



# FP_ENCODE

Encode bit state -> hexadecimal

This FP instruction encodes the contents of data specified by **s1_Start** according to the contents of **s2_Control** if the trigger **EN** is TRUE. The encoded result is stored in the 16-bit destination area specified by **d_Start** starting with the specified bit position. Invalid bits in the area specified for the encoded result are filled with zeros.

**FP_ENCODE** is the inverse instruction of **FP_DECODE**.


## Parameters

Input

**s1_Start** (WORD)

Starting 16-bit area to be encoded (source)

**s2_Control** (WORD)

Control data to specify the starting bit position and number of bits to be encoded

Output

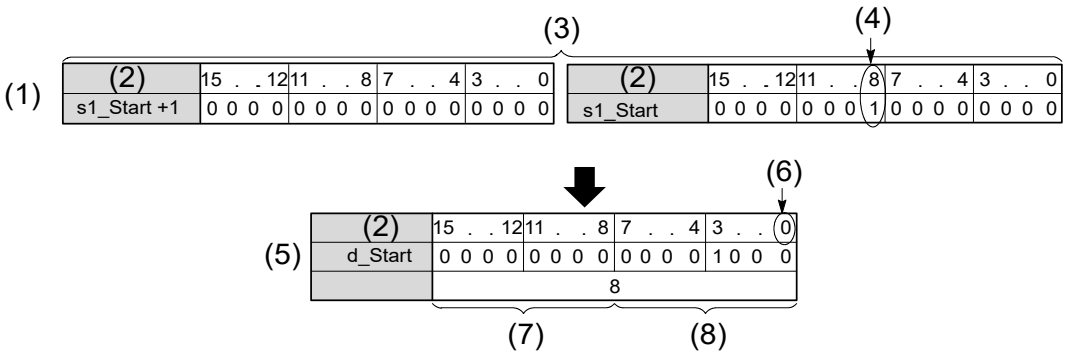
**d_Start** (WORD)

16-bit area for storing encoded data (destination)

## Remarks

- The variables **s1_Start**, **s2_Control** and **d_Start** have to be of the same data type.
- Example with **s2_Control**=16#0005

| | | | | | |
|---|---|---|---|---|---|
| 16# | 0 | 0 | 0 | 5 | |
| | | ↑ | | ↑ | Number of bits to be encoded: $2^5$=32 bits |
| | | Starting bit position of the data to be encoded: bit position 0 | | | |




(1) Source

(2) Bit position

(3) 32 bits as specified by 16#0005

(4) The 8th bit of 32-bit data is TRUE.

(5) Destination
The encoded result 8 (decimal) is stored in **d_Start** when the trigger **EN** is TRUE.

(6) Starting bit position

(7) Bit position 8–15 are filled with zeros.

(8) Encoded result: 8

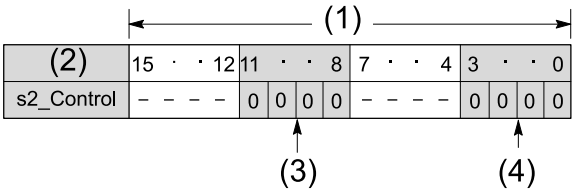
**Description of the process**

The contents of the 2^nL bit segment at the beginning of the area specified by **s1_Start** is encoded (where nL is the number of bits to be encoded).

The encoded result is stored as decimal data in the eight bits starting from the bit specified as the nH bit (where nH is the conversion start bit). Invalid bits in the specified area for the result are filled with zeros.

**Description of s2_Control**

**s2_Control** specifies the starting bit position of the destination data and the number of bits to be encoded using hexadecimal data.




(1) 16-bit data

(2) Number of bit 0–15

The bits marked "-" are invalid.

| Set value 16#0–16#F | (3) Starting bit position of encoded destination data | | Set value 16#0–16#8 | (4) Number of encoded bits |
|---|---|---|---|---|
| 16#0 | 0 | | 16#1 | 2 |
| 16#1 | 1 | | 16#2 | 4 |
| 16#2 | 2 | | 16#3 | 8 (1 byte) |
| 16#3 | 3 | | 16#4 | 16 (1 word) |
| 16#4 | 4 | | 16#5 | 32 (2 words) |

| | |
|---|---|
| 16#5 | 5 |
| 16#6 | 6 |
| 16#7 | 7 |
| 16#8 | 8 |
| 16#9 | 9 |
| 16#A | 10 |
| 16#B | 11 |
| 16#C | 12 |
| 16#D | 13 |
| 16#E | 14 |
| 16#F | 15 |

| | |
|---|---|
| 16#6 | 64 (4 words) |
| 16#7 | 128 (8 words) |
| 16#8 | 256 (16 words) |

**Encoded example**

When encoding 16-bit data (nL=4), the encoded results are as shown below.

| Data to be encoded (4 bits) | Encoded result | |
|---|---|---|
| | Hex. | Dec. |
| 0000 0000 0000 0001 | 16#0000 | 0 |
| 0000 0000 0000 0010 | 16#0001 | 1 |
| 0000 0000 0000 0100 | 16#0010 | 2 |
| 0000 0000 0000 1000 | 16#0011 | 3 |
| 0000 0000 0001 0000 | 16#0100 | 4 |
| 0000 0000 0010 0000 | 16#0101 | 5 |
| 0000 0000 0100 0000 | 16#0110 | 6 |
| 0000 0000 1000 0000 | 16#0111 | 7 |
| 0000 0001 0000 0000 | 16#1000 | 8 |
| 0000 0010 0000 0000 | 16#1001 | 9 |
| 0000 0100 0000 0000 | 16#1010 | 10 |
| 0000 1000 0000 0000 | 16#1011 | 11 |
| 0001 0000 0000 0000 | 16#1100 | 12 |
| 0010 0000 0000 0000 | 16#1101 | 13 |
| 0100 0000 0000 0000 | 16#1110 | 14 |
| 1000 0000 0000 0000 | 16#1111 | 15 |

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

- if 1 ≤ nL ≤ 8, where nL is the number of bits to be encoded/decoded.
- if 1 ≤ nH + nL ≤ 16, where nH is the conversion start bit and nL is the number of bits to be encoded/decoded.
- if the data to be encoded is all zero.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if 1 ≤ nL ≤ 8, where nL is the number of bits to be encoded/decoded.
- if 1 ≤ nH + nL ≤ 16, where nH is the conversion start bit and nL is the number of bits to be encoded/decoded.
- if the data to be encoded is all zero.

## Example

### POU header

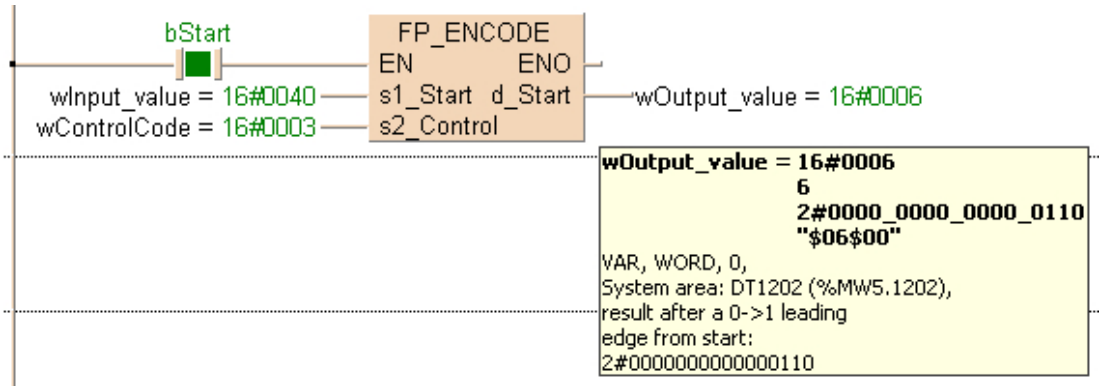
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | wInput_value | WORD | 2#0000000001000000 | |
| 2 | VAR | wControlCode | WORD | 16#0003 | specifies the encodation |
| 3 | VAR | wOutput_value | WORD | 0 | result after a 0->1 leading |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

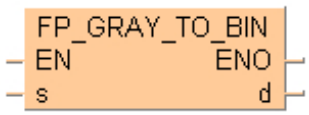
### LD body



# FP_GRAY_TO_BIN

Gray code -> binary data conversion

This FP instruction converts the gray code value at input **s** into binary data if the trigger **EN** is TRUE. The result is stored in **d**.




## Parameters

Input

**s** (WORD, DWORD)

Gray code value

Output

**d** (WORD, DWORD)

Binary data

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## POU header

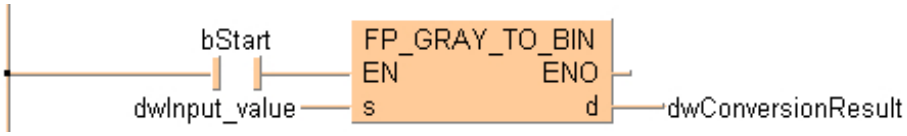
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the fuction |
| 1 | VAR | dwInput_value | DWORD | 16#0000000E | |
| 2 | VAR | dwConversionResult | DWORD | 0 | result after a 0->1 leading |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

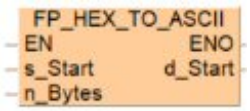
### LD body



# FP_HEX_TO_ASCII

HEX -> ASCII conversion

This FP instruction converts the hexadecimal data at **s_Start** into ASCII codes if the trigger **EN** is TRUE. **n_Bytes** specifies the number of bytes to be converted. The result is stored in the area specified by **d_Start**. ASCII code requires 8 bits (1 byte) to express one hexadecimal character. Upon conversion to ASCII, the data length will thus be twice the length of the source data.




## Parameters

Input

**s_Start** (WORD, DWORD)

Hexadecimal data

**n_Bytes** (INT, DINT, UINT, UDINT)

Number of bytes

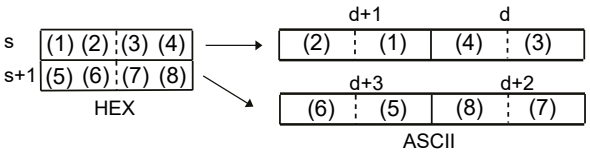
Output

**d_Start** (WORD, INT, UINT)

Starting address of the data area for the results. The size is **n_Bits** * 2 words.

## Remarks

The two characters that make up one byte are interchanged when stored. Two bytes are converted as one segment of data.




## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the byte number specified by **n_Bytes** is greater than the area specified by **s_Start**
- if the calculated result exceeds the area specified by **d_Start**

- if **n_Bytes**= 0 or a negative value

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the byte number specified by **n_Bytes** is greater than the area specified by **s_Start**
- if the calculated result exceeds the area specified by **d_Start**
- if **n_Bytes**= 0 or a negative value

### Example16-bit data

| | Offset | Hex. characters | | | Offset | Converted ASCII codes | String equivalent |
|---|---|---|---|---|---|---|---|
| **s_Start** | 0 | 16#ABCD | ⇒ | **d_Start** | 0 | 16#4443 | DC |
| **n_Bytes** | | 16#0002 | | | | 16#4241 | BA |

### Example32-bit data

| | Offset | Hex. characters | | | Offset | Converted ASCII codes | String equivalent |
|---|---|---|---|---|---|---|---|
| **s_Start** | 0 | 16#1234 | ⇒ | **d_Start** | 0 | 16#3433 | 43 |
| | 1 | 16#5678 | | | 1 | 16#3231 | 21 |
| **n_Bytes** | | 16#0004 | | | | 16#3837 | 87 |
| | | | | | | 16#3635 | 65 |

## Example

### POU header

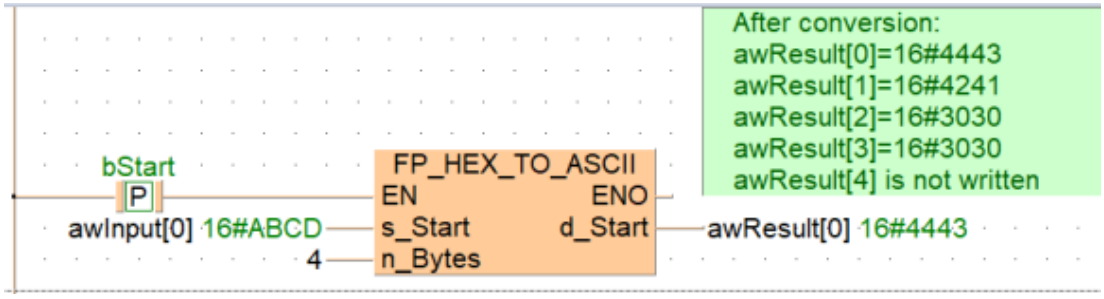
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | | | | |
|---|---|---|---|---|
| 1 | VAR | bStart | BOOL | FALSE |
| 2 | VAR | awInput | ARRAY [0..2] OF WORD | [16#ABCD,2(0)] |
| 3 | VAR | awResult | ARRAY [0..5] OF WORD | [6(16#FFFF)] |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

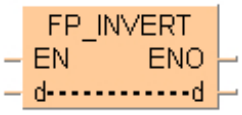
### LD body


After conversion:
awResult[0]=16#4443
awResult[1]=16#4241
awResult[2]=16#3030
awResult[3]=16#3030
awResult[4] is not written
bStart
P
awInput[0] 16#ABCD
4
FP_HEX_TO_ASCII
EN
ENO
s_Start
d_Start
n_Bytes
awResult[0] 16#4443

# FP_INVERT

Data inversion (one's complement)

This FP instruction inverts each bit (0 or 1) of the data area specified by **d** if the trigger **EN** is TRUE. The result is stored in the data area specified by **d**. This instruction is useful for controlling an external device that uses negative logic operation.




## Parameters

Input/output

**d** (WORD, DWORD)

Bit area to be inverted

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## POU header

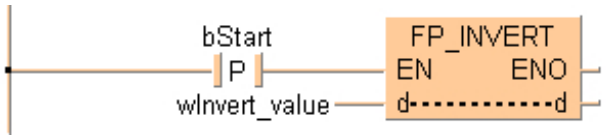
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | wInvert_value | WORD | 2#111000111000 | result after a 0->1 leading |

## POU body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out.

**LD body**



# FP_RAD

Degrees -> radians conversion

This FP instruction converts the value of an angle entered at input **s** from degrees into radians and returns the result at output **d**.


## Parameters

Input

**s** (ANY_REAL)

Angle value in degrees

Output

**d** (ANY_REAL)

Angle value in radians

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s** is not a REAL number

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s** is not a REAL number

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | rInput_value | REAL | 90.0 | corresponds to 90° |
| 2 | VAR | rOutput_value | REAL | 0.0 | result after a 0->1 leading |

## POU body

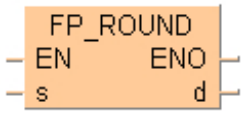
When the variable **start** is set to TRUE, the function is carried out.

## LD body

# FP_ROUND

Round up to the nearest integer

This FP instruction rounds up to the nearest integer.




## Parameters

Input

**s** (ANY_REAL)

Input value

Output

**d** (ANY_REAL)

Result

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s** is not a REAL number

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s** is not a REAL number

## Example

## POU header

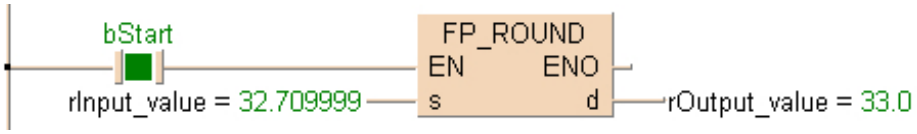
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | rInput_value | REAL | 32.71 | |
| 2 | VAR | rOutput_value | REAL | 0.0 | result after a 0->1 leading |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

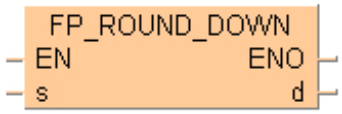
### LD body



# FP_ROUND_DOWN

Round down to the nearest integer

This FP instruction rounds down to the nearest integer.




## Parameters

Input

**s** (ANY_REAL)

Input value

Output

**d** (ANY_REAL)

Result

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s** is not a REAL number

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s** is not a REAL number

## Example

## POU header

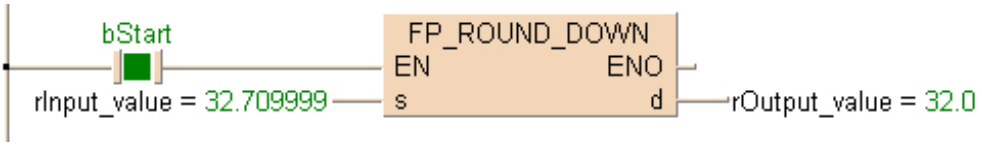
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | rInput_value | REAL | 32.71 | |
| 2 | VAR | rOutput_value | REAL | 0.0 | result after a 0->1 leading |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

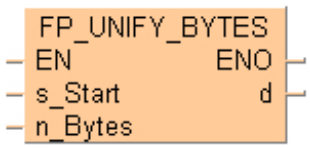
### LD body



# FP_UNIFY_BYTES

Byte combination

This FP instruction unifies the number of bytes specified at **n_Bytes** of the data starting at **s_Start** if the trigger **EN** is TRUE. The result is stored in **d**.




## Parameters

Input

**s_Start** (WORD)

Input value

**n_Bytes** (INT, DINT, UINT, UDINT)

- Number of bytes to be unified, range: 0–65535
- For **n_Bytes**=0, the instruction is not executed
- Unused portions of the memory area are filled with zeros.

Output

**d** (WORD, DWORD)

Result

## Error flags

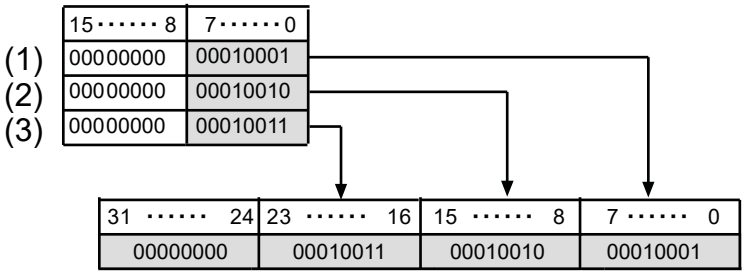
**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the value at **n_Bytes** is out of range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the value at **n_Bytes** is out of range

### Example




(1) Offset 0
(2) Offset 1
(3) Offset 2

## Example

### POU header

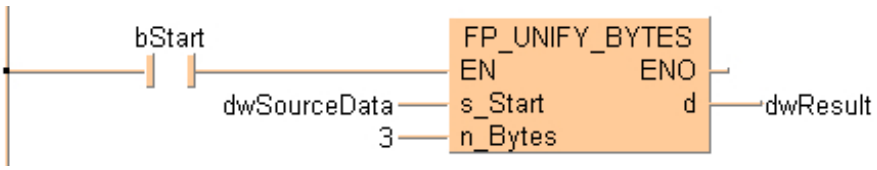
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | dwSourceData | DWORD | 0 |
| 2 | VAR | dwResult | DWORD | 0 |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

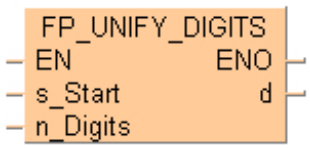
### LD body



# FP_UNIFY_DIGITS

Digit combination

This FP instruction unifies the number of digits specified at **n_Digits** of the data starting at **s_Start** if the trigger **EN** is TRUE. The result is stored in **d**.




## Parameters

Input

**s_Start** (WORD)

Input value, range: 0–65535

**n_Digits** (WORD, INT, UINT)

Number of digits to be unified, range: 0–4
Unused portions of the memory area are filled with zeros.

Output

**d** (REAL)

Result

## Error flags

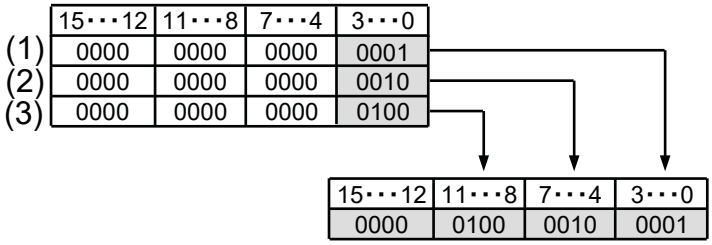
**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the value at **n_Digits** is out of range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the value at **n_Digits** is out of range

### Example




(1) Offset 0
(2) Offset 1
(3) Offset 2

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global variables*

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_GLOBAL | g_arrValuesWORD | ARRAY [0..7] OF WORD | [16#0000,16#0001,16#7FFF,... |

### POU header

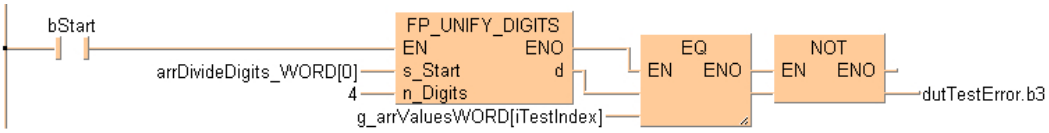
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | iTestIndex | INT | 0 |
| 2 | VAR | dutTestError | BOOL32_OVERLAPPING_DUT | w0 := 0, |
| 3 | VAR | arrDivideDigits_WORD | ARRAY [0..3] OF WORD | [4(0)] |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

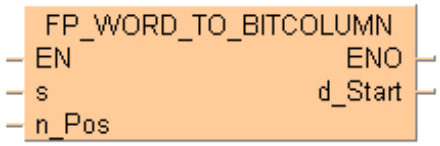
### LD body



# FP_WORD_TO_BITCOLUMN

Bit line -> bit column conversion

This FP instruction creates a bit column out of a value given at input **s** that is returned within the address specified by **d_Start**. The bit position of the column at **d_Start** is specified at input **n_Pos** (range 0–15). This instruction is the reversion of **FP_BITCOLUMN_TO_WORD**.




## Parameters

Input

**s** (WORD)

Input value

**n_Pos** (WORD, INT, UINT)

Bit position (range 0–15)

Output

**d_Start** (WORD)

Starting address of the data area for the results. The size is **n_Bits** * 2 words.; only the specified bit column will be rewritten

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if 0 ≥ **n_Pos** > 15
- if **s_Start** is out of range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if 0 ≥ **n_Pos** > 15
- if **s_Start** is out of range

## Example

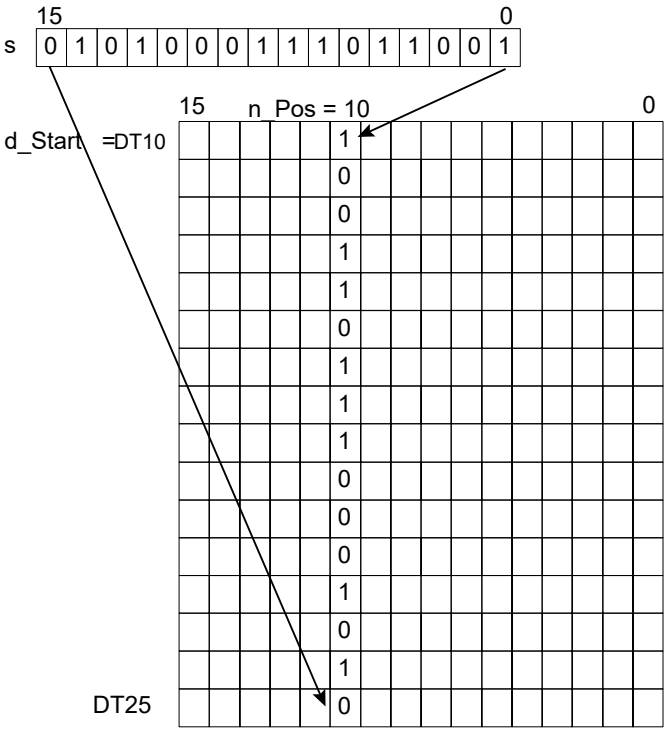
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

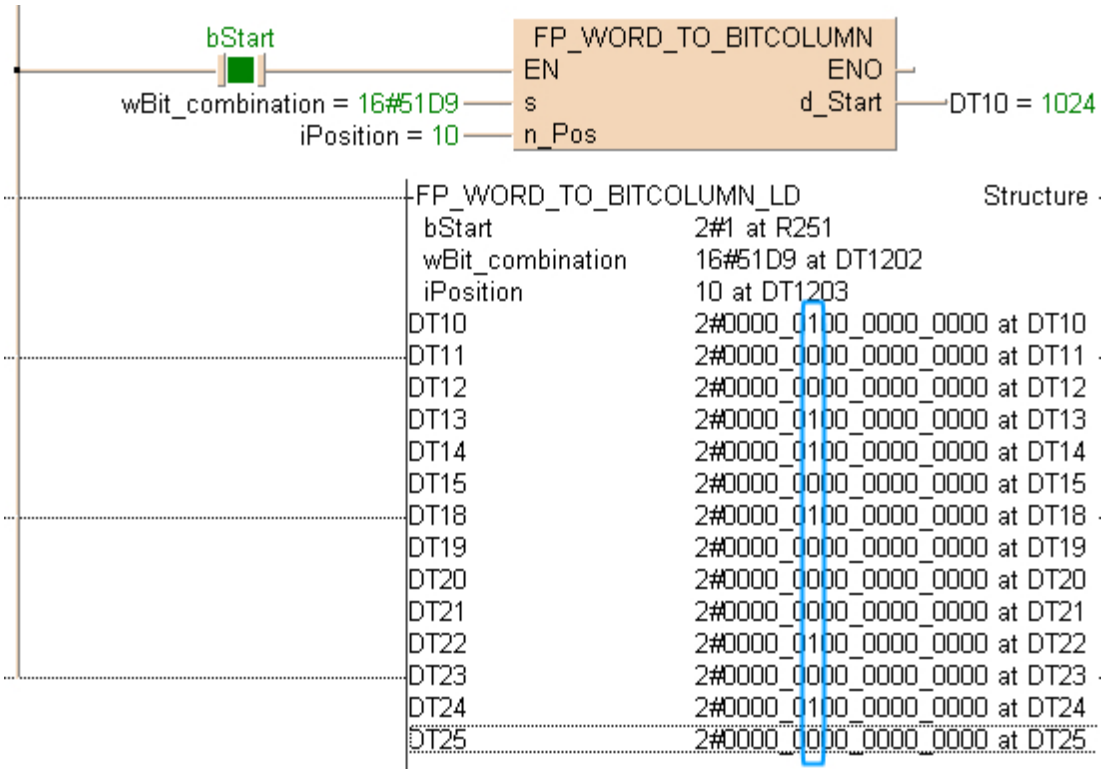
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | wBit_combination | WORD | 16#51D9 | |
| 2 | VAR | iPosition | INT | 10 | specifies the position |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out. The bit line 0101 0001 1101 1001 at input **s** is output at bit position 10 of the data starting at DT10.



### LD body


bStart
FP_WORD_TO_BITCOLUMN
EN
ENO
wBit_combination = 16#51D9
s
d_Start
DT10 = 1024
iPosition = 10
n_Pos
FP_WORD_TO_BITCOLUMN_LD
Structure
bStart
2#1 at R251
wBit_combination
16#51D9 at DT1202
iPosition
10 at DT1203
DT10
2#0000_0100_0000_0000 at DT10
DT11
2#0000_0000_0000_0000 at DT11
DT12
2#0000_0000_0000_0000 at DT12
DT13
2#0000_0100_0000_0000 at DT13
DT14
2#0000_0100_0000_0000 at DT14
DT15
2#0000_0000_0000_0000 at DT15
DT18
2#0000_0100_0000_0000 at DT18
DT19
2#0000_0000_0000_0000 at DT19
DT20
2#0000_0000_0000_0000 at DT20
DT21
2#0000_0000_0000_0000 at DT21
DT22
2#0000_0100_0000_0000 at DT22
DT23
2#0000_0000_0000_0000 at DT23
DT24
2#0000_0100_0000_0000 at DT24
DT25
2#0000_0000_0000_0000 at DT25

## 12.25 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

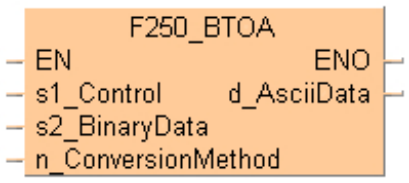
Advantages of IEC instructions

Advantages of FP instructions

# F250_BTOA

Binary -> ASCII conversion

Converts 16-bit/32-bit binary data stored in the area specified by **s2_BinaryData** to ASCII code. The conversion method is specified by **n_ConversionMethod** according to the four control characters of **s1_Control**. The converted result is stored in the area specified by **d_AsciiData**.

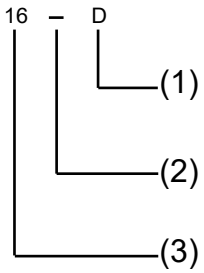



## Parameters

Input

**s1_Control** (STRING)

Control string




(1) D: converts to decimal ASCII data
H: converts to hexadecimal ASCII data

(2) + Normal direction
- Reverse direction

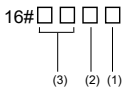
(3) 16: converts in 16-bit (1-word) units
32: converts in 32-bit (2-word) units

**s2_BinaryData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Starting area for storing binary data

**n_ConversionMethod** (WORD, INT, UINT)

Conversion method




(1) Number of ASCII characters per converted unit
(2) Offset in ASCII character units (8-bit)
(3) Number of 16-bit (1-word) or 32-bit (2-word) units converted

(for details, see explanation following the tables)

Output

**d_AsciiData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Starting area for storing ASCII data

## Remarks

Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_BIN_TO_ASCII**

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if there is an error in the control string specified by **s1_Control**.
- if normal direction (+) is specified in **s1_Control** when the format is decimal.
- if the number of ASCII characters per converted unit specified by **n_ConversionMethod** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specifed by **s1_Control**.
- if 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_ConversionMethod**.
- if the number of 16- or 32-bit decimal numbers to be converted specified by **n_ConversionMethod** exceeds the area for storing ASCII data.
- if the converted result exceeds the area.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if there is an error in the control string specified by **s1_Control**.
- if normal direction (+) is specified in **s1_Control** when the format is decimal.
- if the number of ASCII characters per converted unit specified by **n_ConversionMethod** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specifed by **s1_Control**.
- if 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_ConversionMethod**.
- if the number of 16- or 32-bit decimal numbers to be converted specified by **n_ConversionMethod** exceeds the area for storing ASCII data.
- if the converted result exceeds the area.

### Explanation of the conversion method, e.g. n_ConversionMethod = 16#0214

(1) Number of ASCII characters per converted unit (see notes for restrictions)
(2) Register content (hex.)
(3) Character string (see notes)
(4) Offset in ASCII character units (8-bit) for storing the result
(X values do not change)
(5) Number of 16-bit (1-word) or 32-bit (2-word) units to be converted
(6) Register content (hex.)
(7) Values (dec.)

Note

About the number of ASCII characters (8-bit) per converted unit

- When converting 16-bit binary units to hexadecimal ASCII data:
  Range: 16#1–16#4
- When converting 32-bit binary units to hexadecimal ASCII data:
  Range: 16#1–16#8
- When converting binary units to decimal ASCII data:
  Range: 16#1–16#F

## Conversion examples:

- **s1_ Control=**`16+H`, **n_Conversion Method=**`16#204`
  Normal direction. 2 x 4 ASCII characters.

| **Binary data** | | | **Conversion result in ASCII data** | | | |
|---|---|---|---|---|---|---|
| Data type | Offset in 16-bit word units | Hexadecimal number | D | D+1 | D+2 | D+3 |
| INT, WORD | 0 | 16#5678 | '78' | '56' | '34' | '12' |
| | 1 | 16#1234 | | | | |

- **s1_ Control=**`16-H`, **n_Conversion Method=**`16#204`
  Reverse direction. 2 x 4 ASCII characters.

| **Binary data** | | | **Conversion result in ASCII data** | | | |
|---|---|---|---|---|---|---|
| Data type | Offset in 16-bit word units | Hexadecimal number | D | D+1 | D+2 | D+3 |
| INT, WORD | 0 | 16#5678 | '78' | '56' | '12' | '34' |
| | 1 | 16#1234 | | | | |

- **s1_ Control**=16+H, **n_Conversion Method**=16#203
  Normal direction. 2 x 3 ASCII characters.

| **Binary data** | | | **Conversion result in ASCII data** | | | |
|---|---|---|---|---|---|---|
| Data type | Offset in 16-bit word units | Hexadecimal number | D | D+1 | D+2 | D+3 |
| INT, WORD | 0 | 16#0456 | '56' | '42' | '31' | '$xx$xx' |
| | 1 | 16#0123 | | | | |

- **s1_ Control**=16-H, **n_Conversion Method**=16#203
  Reverse direction. 2 x 3 ASCII characters.

| **Binary data** | | | **Conversion result in ASCII data** | | | |
|---|---|---|---|---|---|---|
| Data type | Offset in 16-bit word units | Hexadecimal number | D | D+1 | D+2 | D+3 |
| INT, WORD | 0 | 16#0456 | '45' | '61' | '23' | '$xx$xx' |
| | 1 | 16#0123 | | | | |

- **s1_ Control**=32+H, **n_Conversion Method**=16#108
  Normal direction. 1 x 8 ASCII characters.

| **Binary data** | | | **Conversion result in ASCII data** | | | |
|---|---|---|---|---|---|---|
| Data type | Offset in 16-bit word units | Hexadecimal number | D | D+1 | D+2 | D+3 |
| DINT, DWORD | 0 | 16#12345678 | '78' | '56' | '34' | '12' |

- **s1_ Control**=32-H, **n_Conversion Method**=16#108
  Reverse direction. 1 x 8 ASCII characters.

| **Binary data** | | | **Conversion result in ASCII data** | | | |
|---|---|---|---|---|---|---|
| Data type | Offset in 16-bit word units | Hexadecimal number | D | D+1 | D+2 | D+3 |
| DINT, DWORD | 0 | 16#12345678 | '12' | '34' | '56' | '78' |

- **s1_ Control**=32+H, **n_Conversion Method**=16#105

Normal direction. 1 x 5 ASCII characters.

| Binary data | | | Conversion result in ASCII data | | | |
|---|---|---|---|---|---|---|
| Data type | Offset in 16-bit word units | Hexadecimal number | D | D+1 | D+2 | D+3 |
| DINT, DWORD | 0 | 16#00012345 | '45' | '23' | '1$xx' | '$xx$xx' |

- **s1_ Control**=`32-H`, **n_Conversion Method**=`16#105`

  Reverse direction. 1 x 5 ASCII characters.

| Binary data | | | Conversion result in ASCII data | | | |
|---|---|---|---|---|---|---|
| Data type | Offset in 16-bit word units | Hexadecimal number | D | D+1 | D+2 | D+3 |
| DINT, DWORD | 0 | 16#00012345 | '12' | '34' | '5$xx' | '$xx$xx' |

'x' values do not change.

## Example

### POU header

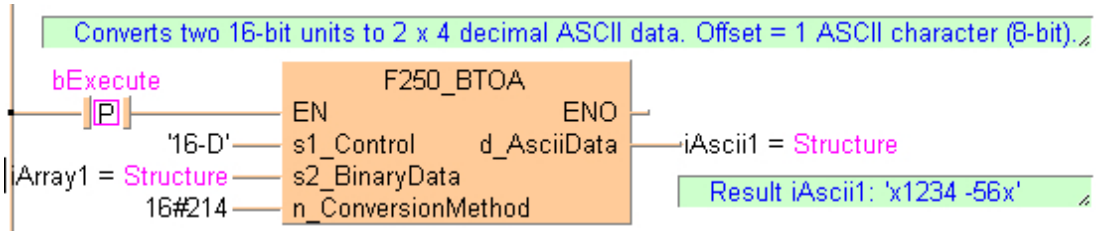
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bExecute | BOOL | FALSE |
| 1 | VAR | iArray1 | ARRAY [0..1] OF INT | [1234,-56] |
| 2 | VAR | iAscii1 | ARRAY [0..4] OF WORD | [5(16#FFFF)] |

### POU body

When **bExecute** is set to TRUE, the instruction is carried out. It converts two 16-bit units to 2 x 4 decimal ASCII data. Offset = 1 ASCII character (8-bit).

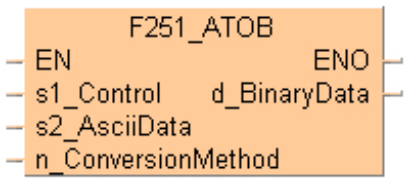
### LD body



# F251_ATOB

ASCII -> binary conversion

Converts ASCII code stored in the area specified by **s2_AsciiData** to 16-bit/32-bit binary data. The conversion method is specified by **n_ConversionMethod** according to the four control characters of **s1_Control**. The converted result is stored in the area specified by **d_BinaryData**.




## Parameters

Input

**s1_Control** (STRING)

Control string, e.g `D-16`

`D` converts decimal ASCII data

`H` converts hexadecimal ASCII data

`+` normal direction

`-` reverse direction

`16` converts the ASCII data to 16-bit data, -32,768 to +32,767 (16#0 to 16#FFFF)

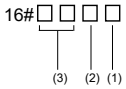
`32` converts the ASCII data to 32-bit data, -2,147,483,648 to +2,147,483,647 (16#0 to 16#FFFFFFFF)

**s2_AsciiData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Starting area for storing ASCII data

**n_ConversionMethod** (WORD, INT, UINT)

Conversion method e.g `16#404`




(1) Number of ASCII characters per converted unit

(2) Offset in ASCII character units (8-bit)

(3) Number of 16-bit (1-word) or 32-bit (2-word) units converted

(for details, see explanation following the tables)

Output

**d_BinaryData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Starting area for storing binary data

## Remarks

Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_ASCII_TO_BIN**

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

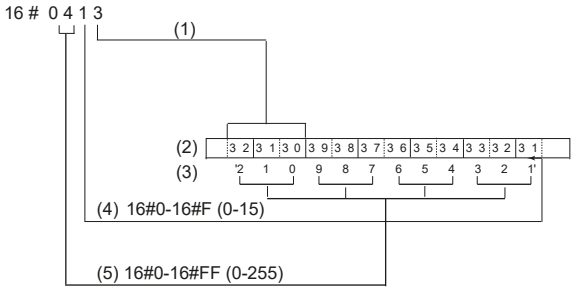
- if there is an error in the control string specified by **s1_Control**.
- if normal direction (+) is specified in **s1_Control** when the format is decimal.
- if the number of ASCII characters per converted unit specified by **n_ConversionMethod** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specifed by **s1_Control**.
- if 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_ConversionMethod**.
- if the number of 16- or 32-bit decimal numbers to be converted specified by **n_ConversionMethod** exceeds the area for storing ASCII data.
- if the converted result exceeds the area.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if there is an error in the control string specified by **s1_Control**.
- if normal direction (+) is specified in **s1_Control** when the format is decimal.
- if the number of ASCII characters per converted unit specified by **n_ConversionMethod** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specifed by **s1_Control**.
- if 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_ConversionMethod**.
- if the number of 16- or 32-bit decimal numbers to be converted specified by **n_ConversionMethod** exceeds the area for storing ASCII data.
- if the converted result exceeds the area.

### Explanation of the conversion method, e.g. n_ConversionMethod = 16#0413 for ASCII data '0123456789012'

For comma-delimited data, specify the maximum number for numeric data. For decimal ASCII data, spaces, symbols and a decimal point are included. The data range is restricted by the control character string **s1_Control**.




(1) Number of ASCII characters per converted unit
(2) Register content (hex.)
(3) Character string
(4) Offset in ASCII character units (8-bit) for storing the result
(5) Number of 16-bit (1-word) or 32-bit (2-word) units to be converted

## Conversion examples for ASCII data '0123456789ABCDEF'

- **s1_ Control**=H+16, **n_ConversionMethod**=16#404
  Normal direction. 4 x 4 ASCII characters.

| ASCII data | | | | | | | | Binary data | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | Data type | Offset in 16-bit word units | Hexadecimal number |
| D | D+1 | D+2 | D+3 | D+4 | D+5 | D+6 | D+7 | INT, WORD | 0 | 16#2301 |
| '01' | '23' | '45' | '67' | '89' | '0B' | 'CD' | 'EF' | | 1 | 16#6745 |
| | | | | | | | | | 2 | 16#AB89 |
| | | | | | | | | | 3 | 16#EFCD |

- **s1_ Control**=H-16, **n_ConversionMethod**=16#404
  Reverse direction. 4 x 4 ASCII characters.

| ASCII data | | | | | | | | Binary data | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | Data type | Offset in 16-bit word units | Hexadecimal number |
| D | D+1 | D+2 | D+3 | D+4 | D+5 | D+6 | D+7 | INT, WORD | 0 | 6#0123 |
| '01' | '23' | '45' | '67' | '89' | '0B' | 'CD' | 'EF' | | 1 | 16#4567 |
| | | | | | | | | | 2 | 16#89AB |
| | | | | | | | | | 3 | 16#CDEF |

- **s1_ Control**=H+16, **n_ConversionMethod**=16#403
  Normal direction. 3 x 4 ASCII characters.

| ASCII data | | | | | | | | Binary data | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | Data type | Offset in 16-bit word units | Hexadecimal number |
| D | D+1 | D+2 | D+3 | D+4 | D+5 | D+6 | D+7 | INT, WORD | 0 | 16#*201 |
| '01' | '23' | '45' | '67' | '89' | '0B' | 'CD' | 'EF' | | 1 | 16#*534 |
| | | | | | | | | | 2 | 16#*867 |
| | | | | | | | | | 3 | 16#*B9A |

- **s1_ Control**=`H-16`, **n_ConversionMethod**=`16#403`
  Reverse direction. 3 x 4 ASCII characters.

| ASCII data | | | | | | | | Binary data | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | Data type | Offset in 16-bit word units | Hexadecimal number |
| D | D+1 | D+2 | D+3 | D+4 | D+5 | D+6 | D+7 | INT, WORD | 0 | 16#*012 |
| '01' | '23' | '45' | '67' | '89' | '0B' | 'CD' | 'EF' | | 1 | 16#*345 |
| | | | | | | | | | 2 | 16#*678 |
| | | | | | | | | | 3 | 6#*9AB |

- **s1_ Control**=`H+32`, **n_ConversionMethod**=`16#802`
  Normal direction. 8 x 2 ASCII characters.

| ASCII data | | | | | | | | Binary data | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | Data type | Offset in 16-bit word units | Hexadecimal number |
| D | D+1 | D+2 | D+3 | D+4 | D+5 | D+6 | D+7 | DINT, DWORD | 0 | 16#67452301 |
| '01' | '23' | '45' | '67' | '89' | '0B' | 'CD' | 'EF' | | 2 | 16#EFCDAB89 |

- **s1_ Control**=`H-32`, **n_ConversionMethod**=`16#802`
  Reverse direction. 8 x 2 ASCII characters.

| ASCII data | | | | | | | | Binary data | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | Data type | Offset in 16-bit word units | Hexadecimal number |
| D | D+1 | D+2 | D+3 | D+4 | D+5 | D+6 | D+7 | DINT, DWORD | 0 | 16#01234567 |
| '01' | '23' | '45' | '67' | '89' | '0B' | 'CD' | 'EF' | | 2 | 16#89ABCDEF |

- **s1_ Control**=`H+32`, **n_ConversionMethod**=`16#502`
  Normal direction. 5 x 2 ASCII characters.

| ASCII data | | | | | | | | Binary data | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | Data type | Offset in 16-bit word units | Hexadecimal number |
| D | D+1 | D+2 | D+3 | D+4 | D+5 | D+6 | D+7 | DINT, DWORD | 0 | 16#***42301 |
| '01' | '23' | '45' | '67' | '89' | '0B' | 'CD' | 'EF' | | 2 | 16#***97856 |

- **s1_ Control**=`H-32`, **n_ConversionMethod**=`16#502`
  Reverse direction. 5 x 2 ASCII characters.

| ASCII data | | | | | | | | Binary data | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | Data type | Offset in 16-bit word units | Hexadecimal number |
| D | D+1 | D+2 | D+3 | D+4 | D+5 | D+6 | D+7 | DINT, DWORD | 0 | 16#***01234 |
| '01' | '23' | '45' | '67' | '89' | '0B' | 'CD' | 'EF' | | 2 | 16#***56789 |

* The extra characters become '0'.

## Example

## POU header

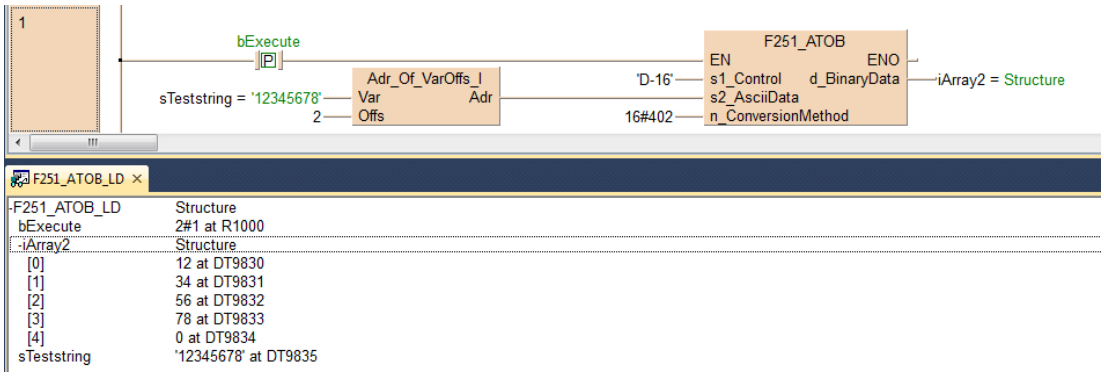
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bExecute | BOOL | FALSE |
| 1 | VAR | iArray2 | ARRAY [0..4] OF INT | [5(0)] |
| 2 | VAR | sTeststring | STRING[32] | '12345678' |

## POU body

When **bExecute** is set to TRUE, the instruction is carried out. It converts 2 x 4 decimal ASCII characters to binary data. Offset = 0 ASCII character (8-bit).

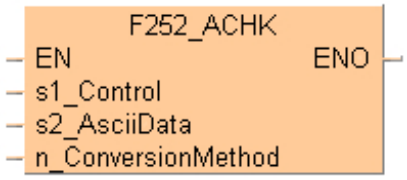
## LD body



# F252_ACHK

ASCII data check

Checks whether the ASCII codes stored in the area specified by **s2_AsciiData** can be converted correctly using the conversion method specified in by **n_ConversionMethod** and the 4 control characters specified by **s1_Control**.

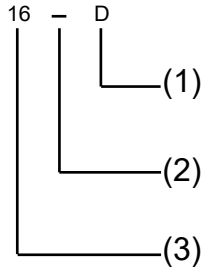



## Parameters

Input

**s1_Control** (STRING)

Control string




(1) D: converts to decimal ASCII data
H: converts to hexadecimal ASCII data

(2) + Normal direction
- Reverse direction

(3) 16: converts in 16-bit (1-word) units
32: converts in 32-bit (2-word) units

**s2_AsciiData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Starting area for storing ASCII data

**n_ConversionMethod** (WORD, INT, UINT)

16-bit equivalent constant or 16-bit area for storing conversion method

Output

**d_BinaryData** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Starting area for storing binary data

## Remarks

Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **F251_ATOB**

## Remarks

- If the results are correct, the system variable **sys_bIsEqual** turns on.
- If the results are incorrect, the system variable **sys_bIsEqual** turns off.
- For an detailed description of **s1_Control** and **F251_ATOB**, please refer to .

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if there is an error in the control string specified by **s1_Control**.
- if normal direction (+) is specified in **s1_Control** when the format is decimal.
- if the number of ASCII characters per converted unit specified by **n_ConversionMethod** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specifed by **s1_Control**.
- if 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_ConversionMethod**.
- if the number of 16- or 32-bit decimal numbers to be converted specified by **n_ConversionMethod** exceeds the area for storing ASCII data.
- if the converted result exceeds the area.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if there is an error in the control string specified by **s1_Control**.
- if normal direction (+) is specified in **s1_Control** when the format is decimal.
- if the number of ASCII characters per converted unit specified by **n_ConversionMethod** exceeds 4 for 16-bit data or 8 for 32-bit data when hexadecimal format is specifed by **s1_Control**.
- if 0 is specified for the no. of 16- or 32-bit (1- or 2-word) units to be converted in **n_ConversionMethod**.
- if the number of 16- or 32-bit decimal numbers to be converted specified by **n_ConversionMethod** exceeds the area for storing ASCII data.
- if the converted result exceeds the area.

## Example

### POU header

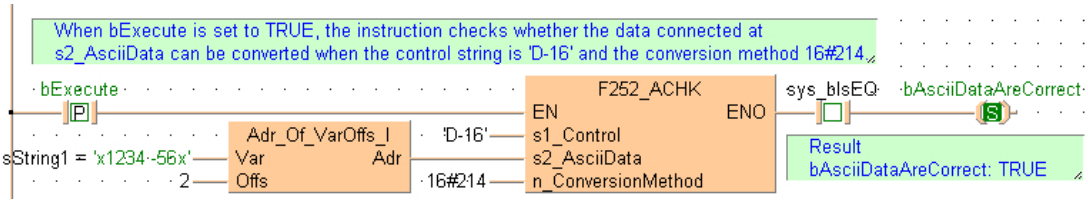
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bExecute | BOOL | FALSE |
| 1 | VAR | sString1 | STRING[10] | '1234567890' |
| 2 | VAR | bAsciiDataAreCorrect | BOOL | FALSE |

### POU body

When **bExecute** is set to TRUE, the instruction checks whether the data connected at **s2_AsciiData** can be converted when the control string is 'D-16' and the conversion method 16#214.

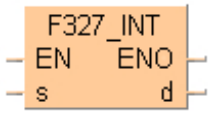
### LD body



# F327_INT

Floating point data -> 16-bit integer data (the largest integer not exceeding the floating point data)

The function converts a floating point data at input **s** in the range -32767.99 to 32767.99 into integer data (including +/- sign). The result of the function is returned at output **d**.




## Parameters

Input

**s** (REAL)

Source REAL number data (2 words)

Output

**d** (INT, WORD)

Destination for storing converted data

## Remarks

The converted integer value at output **d** is always less than or equal to the floating point value at input **s**:

- When there is a positive floating point value at the input, a positive pre-decimal value is returned at the output.
- When there is a negative floating point value at the input, the next smallest pre-decimal value is returned at the output.
- If the floating point value has only zeros after the decimal point, its pre-decimal point value is returned.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the value at input **s** is not a REAL number, or the converted result exceeds the 16-bit area at output **d**.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the value at input **s** is not a REAL number, or the converted result exceeds the 16-bit area at output **d**.

**sys_bIsEqual** (turns to TRUE and remains TRUE)

if the calculated result is 0.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

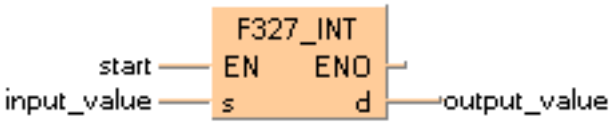
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | input_value | REAL | -1.234 | |
| 2 | VAR | output_value | INT | 0 | result: here -2 |

In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

### POU body

When the variable **start** is set to TRUE, the function is carried out. It converts the floating point value -1.234 into the whole number value -2, which is transferred to the variable **output_value**. Since the whole number may not exceed the floating point value, the function rounds down here.

### LD body



# F328_DINT

Floating point data -> 32-bit integer data (the largest integer not exceeding the floating point data)

The function converts a floating point data at input **s** in the range -2147483000 to 214783000 into integer data (including +/- sign). The result of the function is returned at output **d**.


## Parameters

Input

**s** (REAL)

Source REAL number data (2 words)

Output

**d** (DINT, DWORD)

Destination for storing converted data

## Remarks

The converted integer value at output **d** is always less than or equal to the floating point value at input **s**:

- When there is a positive floating point value at the input, a positive pre-decimal value is returned at the output.
- When there is a negative floating point value at the input, the next smallest pre-decimal value is returned at the output.
- If the floating point value has only zeros after the decimal point, its pre-decimal point value is returned.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the value at input **s** is not a REAL number, or the converted result exceeds the 32-bit area at output **d**.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the value at input **s** is not a REAL number, or the converted result exceeds the 32-bit area at output **d**.

**sys_bIsEqual** (turns to TRUE and remains TRUE)

if the calculated result is 0.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | input_value | REAL | -1234567.89 | |
| 2 | VAR | output_value | DINT | 0 | result: here -1234568 |

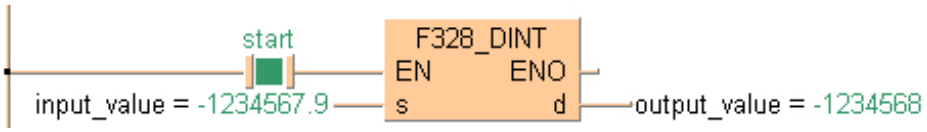
In this example, the input variable **input_value** is declared. However, you can write a constant directly at the input contact of the function instead.

### POU body

When the variable **start** is set to TRUE, the function is carried out.

It converts the floating point value -1234567.89 into the whole number value -1234568, which is transferred to the variable **output_value** at the **output**. Since the whole number may not exceed the floating point value, the function rounds down here.

### LD body

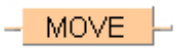


# 13 Copy and initialize instructions

# MOVE

Move value to specified destination

**MOVE** assigns the unchanged value of the input variable to the output variable.




## Parameters

Input

**Unnamed input** (ANY)

source

Output

**Unnamed output** (ANY)

output as input
destination

## Remarks

When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

# Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_value | INT | 0 | all types allowed |
| 1 | VAR | output_value | INT | 0 | all types allowed |

In this example the input variable **input_value** has been declared. Instead, you may enter a constant directly at the input contact of a function.

## POU body

**Input_value** is assigned to **output_value** without being modified.

### LD body

## 13.2 FP instructions

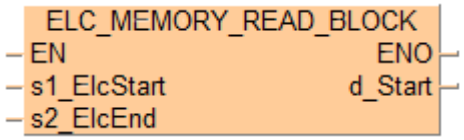
Tip

Advantages of FP instructions

# ELC_MEMORY_READ_BLOCK

Read data block from ELC memory

Read data from the ELC memory of the Eco logic PLC.




Input

**s1_ElcStart** (WORD, INT, UINT)

Starting 16-bit address of the ELC memory area.

**s2_ElcEnd** (WORD, INT, UINT)

Ending 16-bit address of the ELC memory area.

Output

**d_Start** (WORD, INT, UINT)

Starting 16-bit address for storing ELC memory data.

## Example

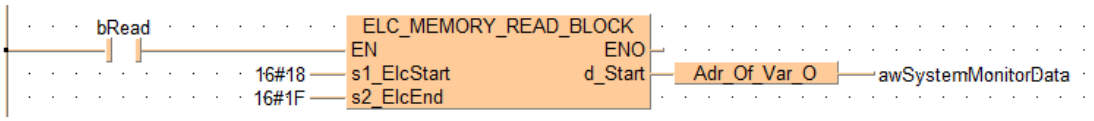
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bRead | BOOL | FALSE |
| 1 | VAR | awSystemMonitorData | ARRAY [0..7] OF WORD | [8(0)] |

### LD body

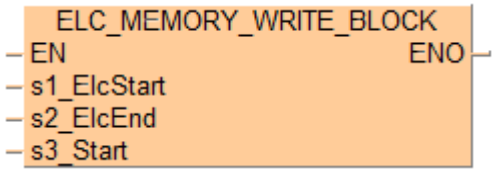
When the variable **bRead** is set to TRUE, the function is carried out.



# ELC_MEMORY_WRITE_BLOCK

Write data block to ELC memory

Write data to the ELC memory of the Eco logic PLC.




Input

**s1_ElcStart** (WORD, INT, UINT)

Starting 16-bit address of the ELC memory area.

**s2_ElcEnd** (WORD, INT, UINT)

Ending 16-bit address of the ELC memory area.

**s3_Start** (WORD, INT, UINT)

Starting 16-bit address of the data to be written to the ELC memory area.

## Example

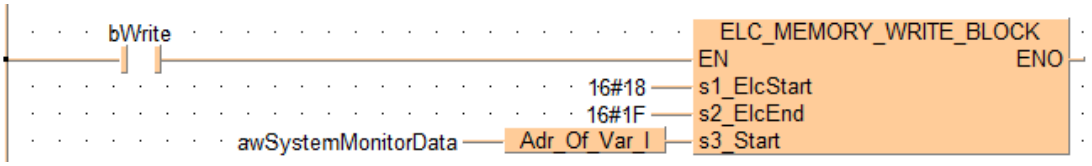
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | awSystemMonitorData | ARRAY [0..7] OF WORD | [8(0)] |
| 1 | VAR | bWrite | BOOL | FALSE |

## LD body

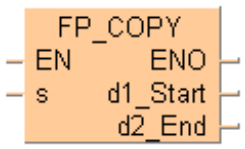
When the variable **bWrite** is set to TRUE, the function is carried out.



# FP_COPY

Copy data to memory block

This FP instruction copies the data or equivalent constant specified by **s** to the memory block specified by **d1_Start** and **d2_End** if the trigger **EN** is TRUE.




## Parameters

Input

**s** (INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT)

Source value to be copied

Output

**d1_Start** (INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT)

Starting address of the destination data area

**d2_End** (INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT)

Ending address of the destination data area

## Remarks

- The input and output variables have to be of the same data type.
- The operands **d1_Start** and **d2_End** should be:
  - in the same memory area, e.g. DT, WR, FL, LD ...
  - **d1_Start** ≤ **d2_End**

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if **d1_Start** > **d2_End**

## Example

### POU header

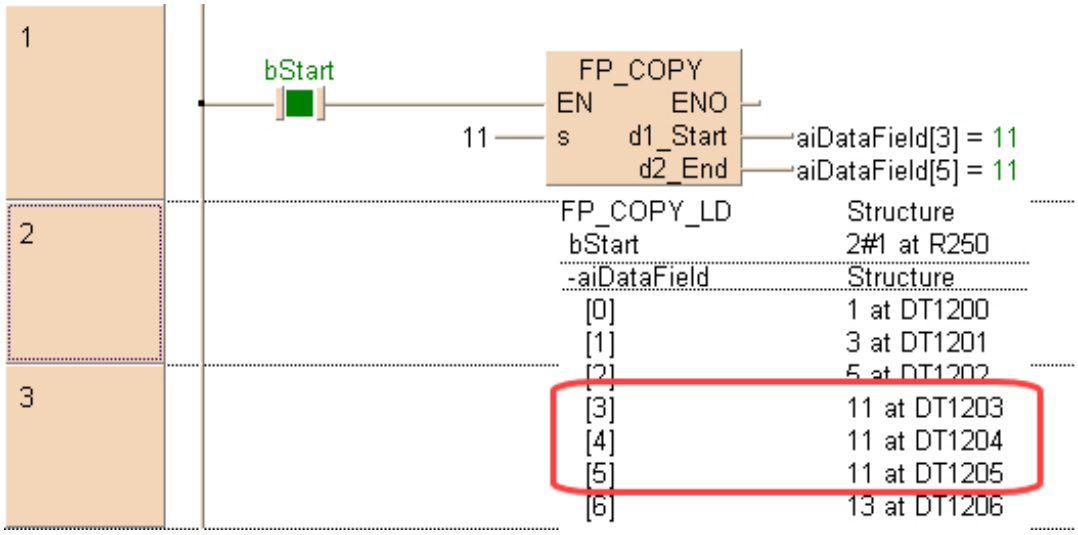
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | aiDataField | ARRAY [0..6] OF INT | [1,3,5,7,9,11,13] | result after a 0->1 leading |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

### LD body



# FP_EXCHANGE

Exchange data

This FP instruction exchanges the contents of the variables connected to **d1** and **d2** if the trigger **EN** is TRUE.


## Remarks

- The input and output variables have to be of the same data type.
- The operands **d1_Start** and **d2_End** should be:
  - in the same memory area, e.g. DT, WR, FL, LD ...
  - **d1_Start** ≤ **d2_End**

## Parameters

Input/output

**d1** (INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT)

Variable to be exchanged with **d2**

**d2** (INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT)

Variable to be exchanged with **d1**

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

### POU header

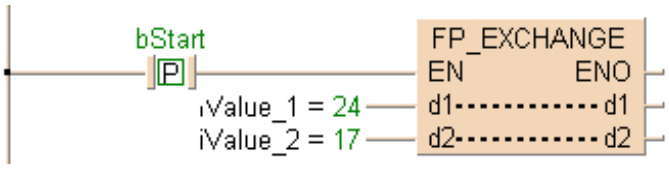
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | iValue_1 | INT | 17 | result after a 0->1 leading |
| 2 | VAR | iValue_2 | INT | 24 | result after a 0->1 leading |

### POU body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out.

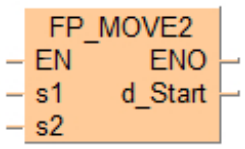
### LD body



# FP_MOVE2

Move two values to a destination area

This FP instruction copies two values from the inputs to a destination area defined by **d_Start**.




## Parameters

Input

**s1**, **s2** (INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT)

Source area

Output

**d_Start** (INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT)

Destination area

## Remarks

The input and output variables have to be of the same data type.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- If the values specified using the index modifier are out of range.
- If the transfer range is outside the accessible range.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- If the values specified using the index modifier are out of range.
- If the transfer range is outside the accessible range.

## Example

### POU header

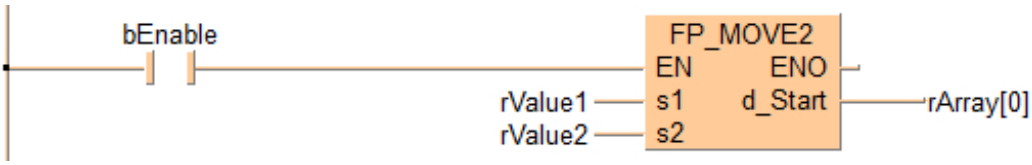
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | rArray | ARRAY [0..1] OF REAL | [2(0.0)] |
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | rValue2 | REAL | 3.4 |
| 3 | VAR | rValue1 | REAL | 1.2 |

### POU body

When the variable **bEnable** is set to TRUE, the function is executed.

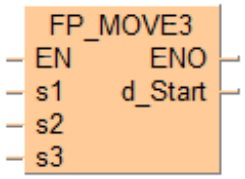
### LD body



# FP_MOVE3

Move three values to a destination area

This FP instruction copies three values to a destination area defined by **d_Start**.




## Parameters

Input

**s1**, **s2**, **s3** (INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT)

Source area

Output

**d_Start** (INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT)

Destination area

## Remarks

The input and output variables have to be of the same data type.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- If the values specified using the index modifier are out of range.
- If the transfer range is outside the accessible range.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- If the values specified using the index modifier are out of range.
- If the transfer range is outside the accessible range.

## Example

### POU header

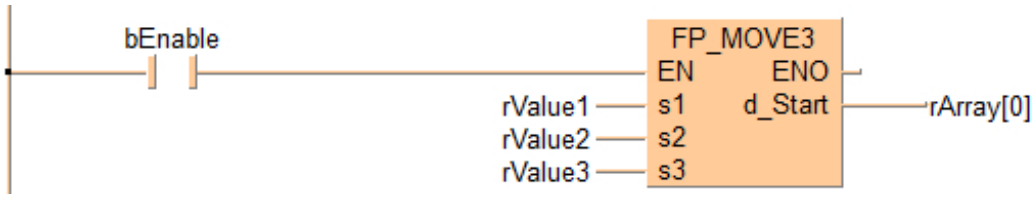
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | rArray | ARRAY[0..2] OF REAL | [3(0.0)] |
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | rValue3 | REAL | 5.6 |
| 3 | VAR | rValue2 | REAL | 3.4 |
| 4 | VAR | rValue1 | REAL | 1.2 |

### POU body

When the variable **bEnable** is set to TRUE, the function is executed.

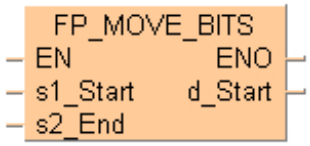
### LD body



# FP_MOVE_BITS

Move bit data

This FP instruction moves a bit data from the bit data area specified by **s1_Start** and **s2_End** to the bit data block starting from the bit data area specified by **d_Start** if the trigger **EN** is TRUE.




## Parameters

Input

**s1_Start** (BOOL)

Starting address of the source bit data area

**s2_End** (BOOL)

Ending address of the source bit data area

Output

**d_Start** (BOOL)

Starting address of the destination bit data area

## Remarks

- The input and output variables have to be of the same data type.
- The operands **s1_Start** and **s2_End** should be:
  - in the same memory area, e.g. DT, WR, FL, LD ...
  - **s1_Start** ≤ **s2_End**

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if **s1_Start** > **s2_End**

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global variables

| | Class | Identifier | FP address | IEC address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | bStartAddress | X1 | %IX0.1 | BOOL | FALSE |
| 1 | VAR_GLOBAL | bEndAddress | X8 | %IX0.8 | BOOL | FALSE |
| 2 | VAR_GLOBAL | bTargetAddress | YD | %QX0.13 | BOOL | FALSE |

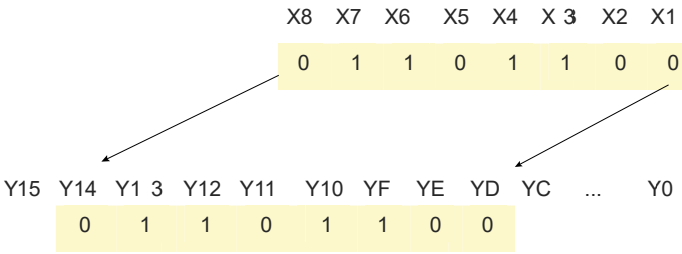
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

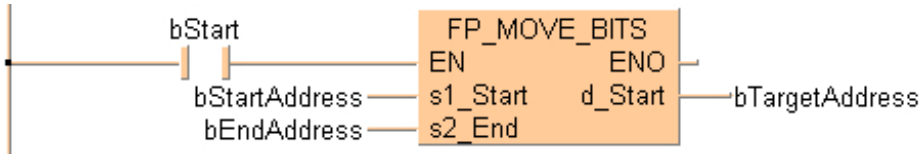
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR_EXTERNAL | bStartAddress | BOOL | FALSE | |
| 2 | VAR_EXTERNAL | bEndAddress | BOOL | FALSE | |
| 3 | VAR_EXTERNAL | bTargetAddress | BOOL | FALSE | |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out. It moves the bits X1–X8 to the target area YD–Y14.




### LD body



# FP_MOVE_BLOCK

Move data block

This FP instruction moves the data block from the data area specified by **s1_Start** and by **s2_End** to the block starting from the data area specified by **d_Start** if the trigger **EN** is TRUE.


## Parameters

Input

**s1_Start** (INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT)

Starting address of the source data area

**s2_End** (INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT)

Ending address of the source data area

Output

**d_Start** (INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT)

Starting address of the destination data area

## Remarks

- The following conditions apply:
  - The input and output variables have to be of the same data type.
  - The operands **s1_Start** and **s2_End** have to be in the same memory area, e.g. DT, WR, FL, LD ...
  - **s1_Start≤s2_End**
- Whenever **s1_Start**, **s2_End** and **d_Start** are in the same data area:
  - **s1_Start=d_Start**: The instruction will not be executed.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start** > **s2_End**

- if the destination range is out of the available range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start** > **s2_End**
- if the destination range is out of the available range

## Example

### POU header

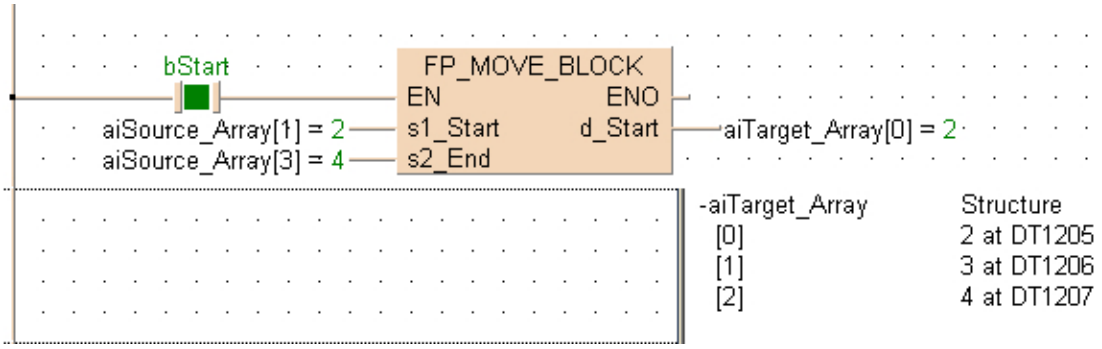
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | aiSource_Array | ARRAY [0..4] OF INT | [1,2,3,4,5] | |
| 2 | VAR | aiTarget_Array | ARRAY [0..2] OF INT | [3(0)] | result after a 0->1 leading edge |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

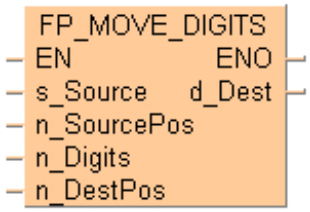
### LD body



# FP_MOVE_DIGITS

Digit data move

This FP instruction moves **n_Digits** number of digits of the source value **s_Source** from digit position **n_SourcePos** to the destination digit position **n_DestPos** of the 16-bit data specified by **d_Dest**.




## Parameters

Input

**s_Source** (WORD)

Data area of source

**n_SourcePos** (INT, DINT, UINT, UDINT)

Specifies source hexadecimal digit position (0–3)

**n_Digits** (INT, DINT, UINT, UDINT)

Number of hexadecimal digits (1–4)

**n_DestPos** (INT, DINT, UINT, UDINT)

Specifies destination hexadecimal digit position (0–3)

Output

**d_Dest** (WORD)

data area of destination

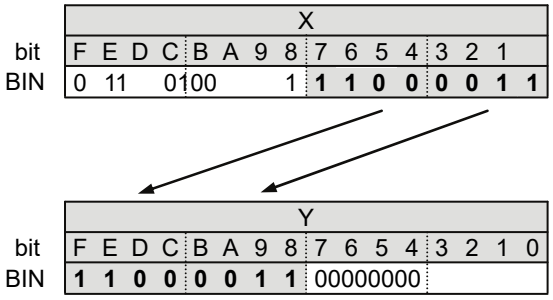
## Remarks

- If the number of digits from the starting digit position exceeds the maximum number of digits (**n_SourcePos** + **n_Digits** > 4), the remaining digits are shifted to the digit position starting from 0 of the same word.
- The operands **s1_Start** and **s2_End** should be:
  - in the same memory area, e.g. DT, WR, FL, LD ...
  - **s1_Start** ≤ **s2_End**

### Example

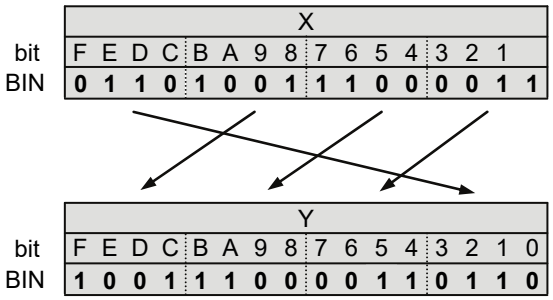
- Move multiple digits

  **s_Source**...WX0
  **n_Digits**...U2
  **d_Dest**...WY0
  **n_SourcePos**...U0
  **n_DestPos**...U2




- Move four digits

  **s_Source**...WX0
  **n_Digits**...U4
  **d_Dest**...WY0
  **n_SourcePos**...U0
  **n_DestPos**...U4




## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if one of the input variables/output variable is out of range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if one of the input variables/output variable is out of range

## Example

### POU header

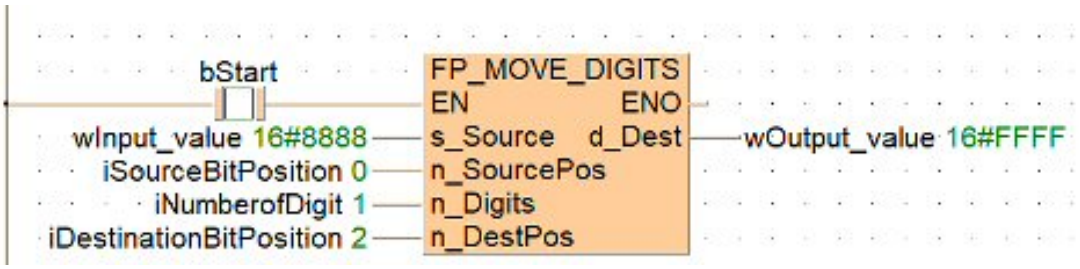
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | wInput_value | WORD | 2#1000100010001000 | |
| 2 | VAR | iSourceBitPosition | INT | 0 | digit no.0 locates the position of the source bit (here: 2) |
| 3 | VAR | iNumberofDigit | INT | 1 | |
| 4 | VAR | iDestinationBitPosition | INT | 2 | digit no.2 locates the position of the destination bit (here: 15) |
| 5 | VAR | wOutput_value | WORD | 2#1111111111111111 | result after a 0->1 leading |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

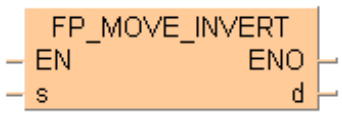
### LD body



# FP_MOVE_INVERT

Data inversion and move

This FP instruction logically inverts the data or equivalent constant specified by **s** and transfers it to the data area specified by **d** if the trigger **EN** is TRUE.




## Parameters

Input

**s** (WORD, DWORD)

Source data to be inverted

Output

**d** (WORD, DWORD)

Destination data

## Remarks

The variables **s** and **d** have to be of the same data type.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## POU header

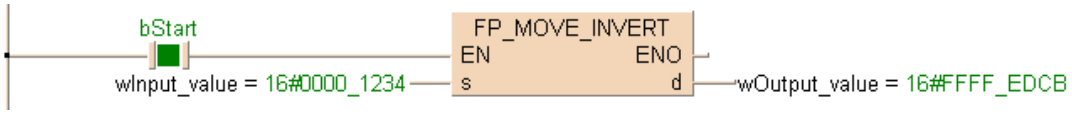
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | wInput_value | DWORD | 16#00001234 | this value will be |
| 2 | VAR | wOutput_value | DWORD | 0 | result after a 0->1 leading |

## POU body

When the variable **bStart** is set to TRUE, the function is carried out.

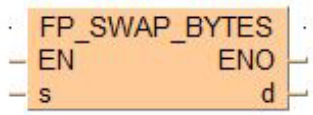
## LD body



# FP_SWAP_BYTES

Exchange higher bytes and lower bytes

This FP instruction exchanges the higher bytes and lower bytes of the data area specified by **s** if the trigger **EN** is TRUE. The result is transferred to **d**.




## Parameters

Input

**s** (WORD)

Source data area

Output

**d** (WORD)

Destination data area

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## POU header

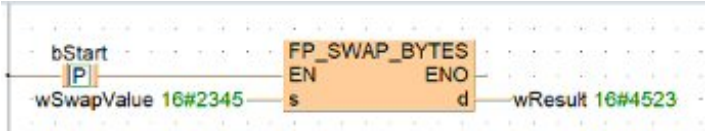
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 1 | VAR | bStart | BOOL | FALSE | activates the function |
| 2 | VAR | wSwapValue | WORD | 16#2345 | result after 0->1 leading edge from start: 16#4523 |
| 3 | VAR | wResult | WORD | 0 | |

### POU body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out.

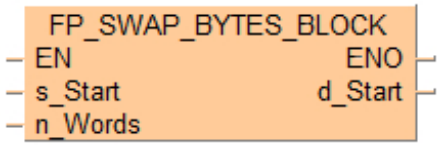
### LD body



# FP_SWAP_BYTES_BLOCK

Exchange higher bytes and lower bytes

This FP instruction exchanges the higher bytes with the lower bytes of all words within the data area specified by **s_Start** and **n_Words** if the trigger **EN** is TRUE. The result is transferred to **d_Start**.




## Parameters

Input

**s_Start** (WORD)

Starting address of the source data area

**n_Words** (WORD, INT, UINT)

Number of words (1–65535, in case of 0 no operation is executed)

Output

**d_Start** (WORD)

Starting address of the destination data area

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## POU header

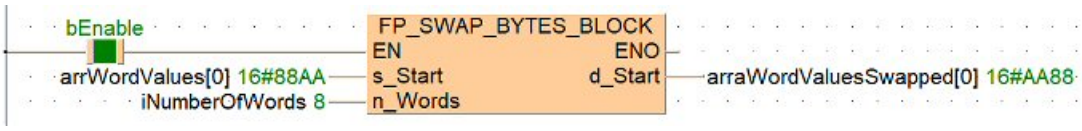
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | arrWordValues | ARRAY [0..7] OF WORD | [8(16#88AA)] |
| 1 | VAR | arraWordValuesSwapped | ARRAY [0..7] OF WORD | [8(0)] |
| 2 | VAR | bEnable | BOOL | FALSE |
| 3 | VAR | iNumberOfWords | INT | 8 |

## POU body

When the variable **bEnable** is set to TRUE, the function is executed.

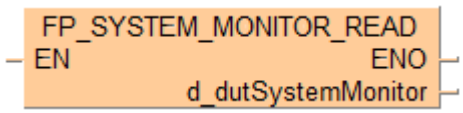
## LD body



# FP_SYSTEM_MONITOR_READ

Reads the system monitor area of the FP7

This FP instruction reads specific areas of the FP7 system monitor and writes the value into a DUT.




## Parameters

Output

**d_dutSystemMonitor** (ANY_DUT)

Use the following predefined DUT:

- FP_SYSTEM_MONITOR_COMMUNICATION_CYCLE_DUT
- FP_SYSTEM_MONITOR_COM_CPU_ERROR_DUT
- FP_SYSTEM_MONITOR_CPU_ERROR_DUT
- FP_SYSTEM_MONITOR_CURRENT_DUT
- FP_SYSTEM_MONITOR_ETHERNET_DUT
- FP_SYSTEM_MONITOR_INTERRUPT_UNITS_DUT
- FP_SYSTEM_MONITOR_LOGGING_TRACE_DUT
- FP_SYSTEM_MONITOR_OPERATION_ERROR_DUT
- FP_SYSTEM_MONITOR_POWER_SUPPLY_DUT
- FP_SYSTEM_MONITOR_SD_CARD_DUT
- FP_SYSTEM_MONITOR_SYNTAX_ERROR_DUT
- FP_SYSTEM_MONITOR_TEMPERATURE_DUT
- FP_SYSTEM_MONITOR_UNIT_ALARM_DUT
- FP_SYSTEM_MONITOR_UNIT_ERROR_DUT
- FP_SYSTEM_MONITOR_UNIT_WARNING_DUT
- FP_SYSTEM_MONITOR_USE_HISTORY_DUT
- FP_SYSTEM_MONITOR_VERIFY_ERROR_DUT
- FP_SYSTEM_MONITOR_VOLTAGE_DUT

## Example

### POU header

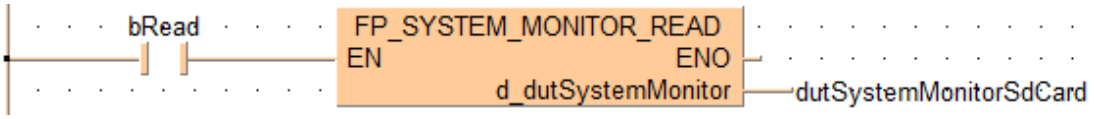
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | dutSystemMonitorSdCard | FP_SYSTEM_MONITOR_SD_CARD_DUT | |
| 1 | VAR | bRead | BOOL | FALSE |

### POU body

When the variable **bRead** is set to TRUE, the function is carried out.

### LD body



## 13.3 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

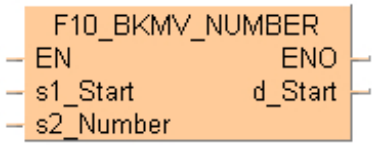
Advantages of IEC instructions

Advantages of FP instructions

# F10_BKMV_NUMBER

Block move by number

The data block specified by the 16-bit starting area specified by **s1_Start** and the number of WORDs specified by **s2_Number** are copied to the block starting from the 16-bit area specified by **d_Start** if the trigger **EN** is in the ON-state.




## Parameters

Input

**s1_Start** (WORD, INT, UINT)

starting 16-bit area, source

**s2_Number** (WORD, INT, UINT)

number of words to be copied, source

Output

**d_Start** (WORD, INT, UINT)

starting 16-bit area, destination

## Remarks

- This instruction is a modification of **F10_BKMV** generated by the compiler.
- Whenever **s1_Start** and **d_Start** are in the same data area:
  - **s1_Start** = **d_Start**: data will be recopied to the same data area.
- The value for '**s2_Number**' has to be greater than 0.
- The variables **s1_Start**, **s2_Number** and **d_Start** have to be of the same data type.

## Example

## POU header

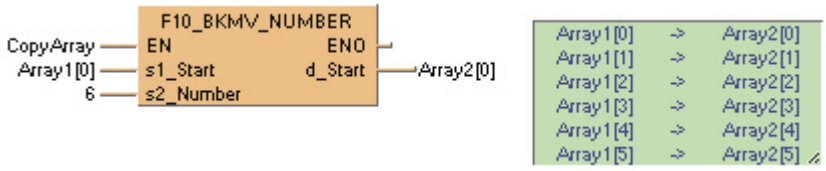
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Array1 | ARRAY [0..5] OF INT | [6(0)] |
| 1 | VAR | Array2 | ARRAY [0..5] OF INT | [6(0)] |
| 2 | VAR | CopyArray | BOOL | FALSE |

## POU body

When the variable **CopyArray** changes from FALSE to TRUE, the function is carried out. It copies the data block starting at the 16-bit area specified by **s1_Start** and the number of WORDs specified by **s2_Number** to the block starting from the 16-bit area specified by **d_Start**.

## LD body

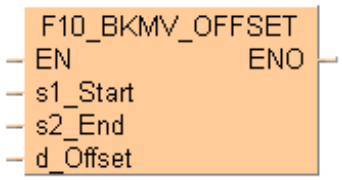


# F10_BKMV_OFFSET

Block move to a offset from source

This instruction is a modification of the **F10_BKMV** (page ) generated by the compiler.

The data block specified by the 16-bit starting area specified by **s1_Start** and 16-bit ending area specified by **s2_End** are copied to the block starting from the 16-bit area specified by the offset **d_Offset** from **s1_Start** if the trigger **EN** is in the ON-state.




## Parameters

Input

**s1_Start** (WORD, INT, UINT)

starting 16-bit area, source

**s2_End** (WORD, INT, UINT)

ending 16-bit area, source

**d_Offset** (WORD, INT, UINT)

offset from s1_Start, destination

## Remarks

- Whenever **s1_Start** and **s2_End** are in the same data area:
  - **d_Offset** = 0: data will be recopied to the same data area.
- The variables **s1_Start**, **s2_End** and **d_Offset** have to be of the same data type.

## Example

## POU header

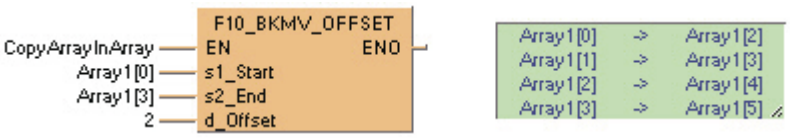
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Array1 | ARRAY [0..5] OF INT | [6(0)] |
| 1 | VAR | CopyArrayInArray | BOOL | FALSE |

## POU body

When the variable **CopyArrayInArray** changes from FALSE to TRUE, the function is carried out. It copies the data block starting at the 16-bit area specified by **s1_Start** and 16-bit ending area specified by **s2_End** to the block starting from the 16-bit area specified by the offset **d_Offset** from **s1_Start**.

## LD body

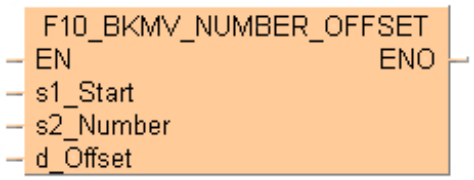


# F10_BKMV_NUMBER_OFFSET

Block move by number to a offset from source

This instruction is a modification of the **F10_BKMV** generated by the compiler.

The data block specified by the 16-bit starting area specified by **s1_Start** and the number of WORDs specified by **s2_Number** are copied to the block starting from the 16-bit area specified by the offset **d_Offset** from **s1_Start** if the trigger **EN** is in the ON-state.




## Parameters

Input

**s1_Start** (WORD, INT, UINT)

starting 16-bit area, source

**s2_Number** (WORD, INT, UINT)

Number of words to be copied, source

**d_Offset** (WORD, INT, UINT)

starting 16-bit area, destination

## Remarks

- Whenever **d_Offset** = 0: data will be recopied to the same data area.
- The value for **s2_Number** has to be greater than 0.
- The variables **s1_Start**, **s2_Number** and **d_Offset** have to be of the same data type.

## Example

## POU header

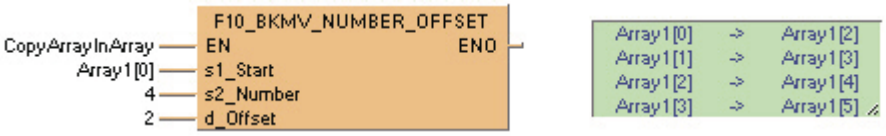
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Array1 | ARRAY [0..5] OF INT | [6(0)] |
| 1 | VAR | CopyArrayInArray | BOOL | FALSE |

## POU body

When the variable **CopyArrayInArray** changes from FALSE to TRUE, the function is carried out. It copies the data block starting at the 16-bit area specified by **s1_Start** and the number of WORDs specified by **s2_Number** to the block starting from the 16-bit area specified by the offset **d_Offset** from **s1_Start**.

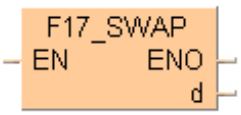
## LD body



# F17_SWAP

Higher/lower byte in 16-bit data exchange

The higher byte (higher 8-bits) and lower bytes (lower 8-bits) of a 16-bit area specified by **d** are exchanged if the trigger **EN** is in the ON-state. 1 byte means 8 bit.




## Parameters

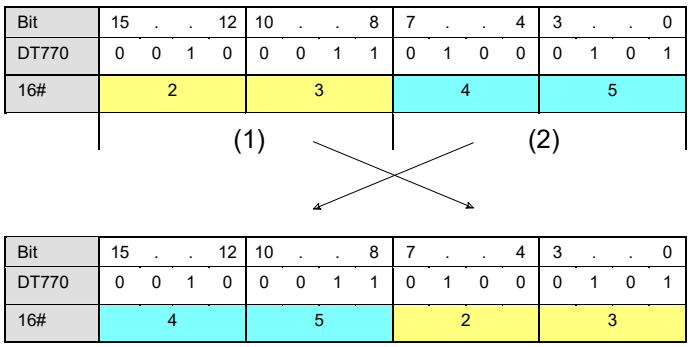
Output

**d** (WORD, INT, UINT)

16-bit area in which the higher and lower bytes are swapped (exchanged)

## Remarks

Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_SWAP_BYTES**




(1) higher byte (8-bit)
(2) lower byte (8-bit)

## Example

## POU header

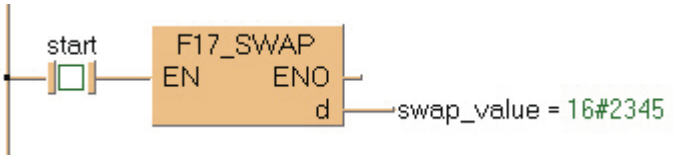
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | swap_value | WORD | 16#2345 | result after 0->1 leading edge from start: 16#4523 |
| 2 | VAR | | | | |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

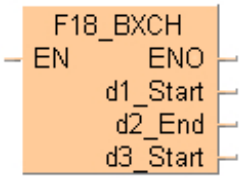
### LD body



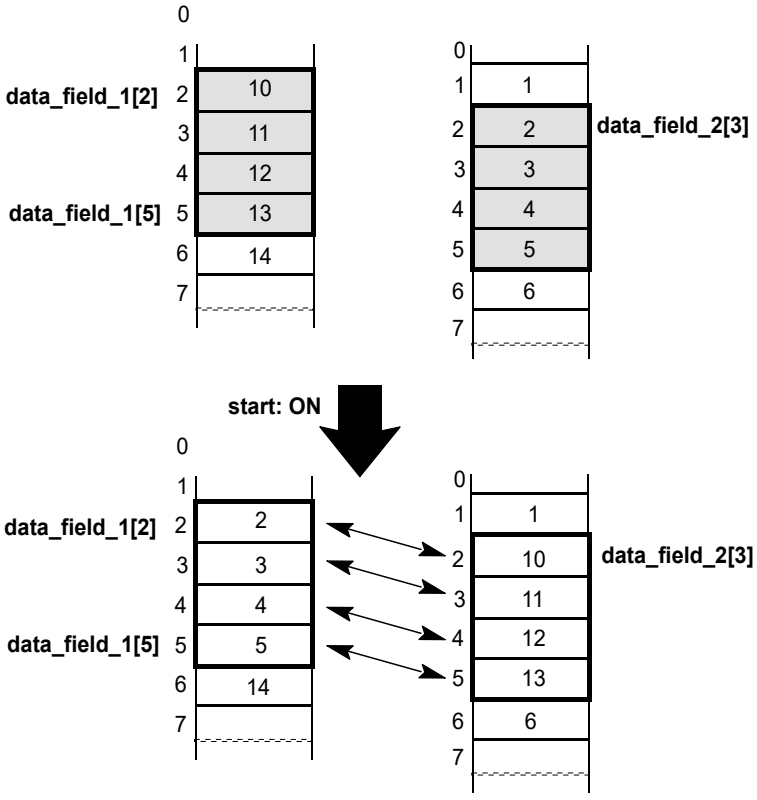
# F18_BXCH

16-bit blocked data exchange

The function exchanges one 16-bit data block for another. The beginning of the first data block is specified at output **d1_Start** and its end at output **d2_End**. Output **d3_Start** specifies the beginning of the second data block.




## Example




## Parameters

Output

**d1_Start** (WORD, INT, UINT)

starting 16-bit area of block data 1

**d2_End** (WORD, INT, UINT)

ending 16-bit area of block data 1

**d3_Start** (WORD, INT, UINT)

starting 16-bit area of block data 2

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the address of the variables at outputs **d1_Start** > **d2_End**
- if the data block to be exchanged is larger than the target area.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the address of the variables at outputs **d1_Start** > **d2_End**
- if the data block to be exchanged is larger than the target area.

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

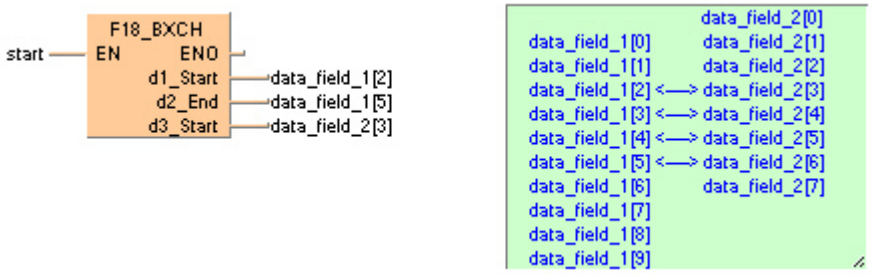
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | data_field_1 | ARRAY [0..9] OF INT | [8,9,10,11,12,13,14,15,16,17] | Arbitrarily large data field |
| 2 | VAR | data_field_2 | ARRAY[0..7] OF INT | [-1,0,1,2,3,4,5,6] | Arbitrarily large data field |

## POU body

When the variable **start** is set to TRUE, the function is carried out.

It exchanges the data of ARRAY **data_field_1** (from the 2$^{nd}$ to the 5$^{th}$ element) with the data of ARRAY **data_field_2** (from the 3$^{rd}$ element on).

### LD body




## 13.4 Data transfer to and from special data registers

To access special data registers and special internal flags, use the PLC-independent system variables.

### 13.4.1 Accessing the HSC special data registers

In this example the system variables of the high-speed counter are called in a ladder diagram. The numbers of data registers differ for the PLC type used.

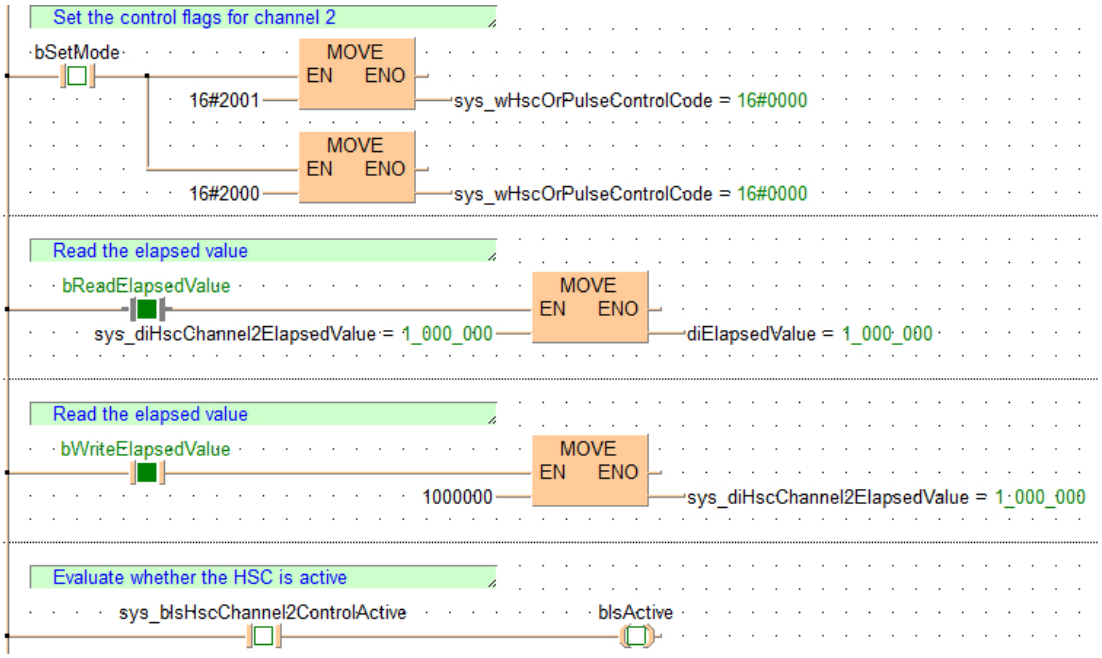
| System variable | FP0 (2,7k, 5k) | FP0 (10k) | FP-Σ | FP-X |
|---|---|---|---|---|
| **sys_wHscOrPulseControlCode** | DT9052 | DT90052 | DT90052 | DT90052 |
| **sys_diHscChannel2ElapsedValue** | DT9104 | DT90104 | DT90200 | DT90308 |
| **sys_bIsHscChannel2ControlActive** | R903C | R903C | R903C | R9112 |

### POU header

All input and output variables used for programming this function have been declared in the POU header.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bReadElapsedValue | BOOL | FALSE |
| 1 | VAR | bWriteElapsedValue | BOOL | FALSE |
| 2 | VAR | bSetMode | BOOL | FALSE |
| 3 | VAR | diElapsedValue | DINT | 0 |
| 4 | VAR | bIsActive | BOOL | FALSE |

### LD body




## 13.4.2 Accessing the RTC special data registers

In this example the system variables of the real time clock are called in structured text. The numbers of data registers differ for the PLC type used.

| System variable | FP0, FPe | FPΣ, FP-X, FP2, FP2SH |
|---|---|---|
| **sys_wClockCalendarMinSec** | DT9054 | DT90054 |
| **sys_wClockCalendarDayHour** | DT9055 | DT90055 |
| **sys_wClockCalendarYearMonth** | DT9056 | DT90056 |
| **sys_wClockCalendarSet** | DT9058 | DT90058 |

### POU header

In the POU header, all input and output variables are declared that are used for programming this function.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetNewRTC | BOOL | FALSE |

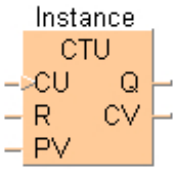
# 14 Counter instructions

# CTU

Up counter

The function block **CTU** (count up) allows you to program counting procedures.

Counting up starts at zero until the maximum value 32767 is reached. Each rising edge at count up **CU** increases the value at current value **CV** by 1.




## Parameters

Input

**CU** (BOOL)

count up
the value 1 is added to **CV** for each rising edge at **CU**, except when **R** is set

**R** (BOOL)

Reset
A rising edge at **R** resets the currrent value **CV** to zero and counting stops. The output **Q** is set to FALSE.
The next falling edge at **R** restarts counting.

**PV** (INT)

Preset value
if **CV** is equal/greater than **PV**, **Q** is set to TRUE
If no preset value is set or the preset value is zero, the output **Q** is set to TRUE immediately after counting starts.

Output

**Q** (BOOL)

signal output
is set to TRUE if **CV** is greater than/equal to **PV**
is set to FALSE if a rising edge is detected at **R**

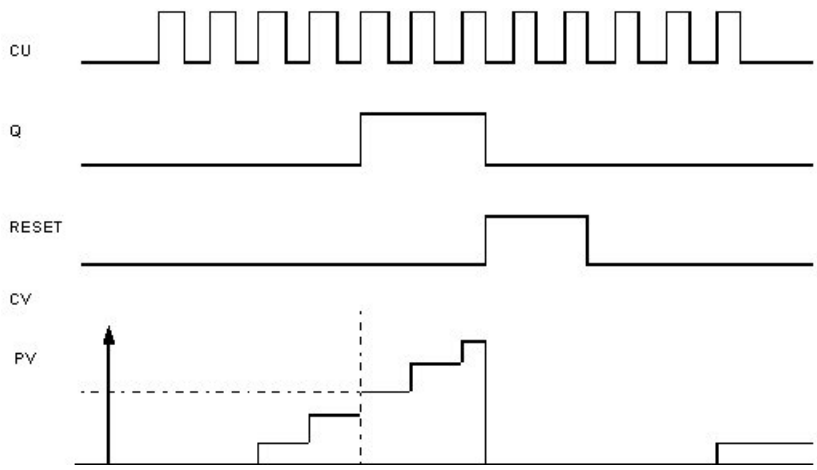
**CV** (INT)

Current value

- contains the addition result

- If **CV** reaches the preset value **PV**, the output **Q** is set to TRUE, but counting continues until the maximum limit 32767 is reached.
- The value can be changed during counting operation by writing to the variable from the programming editor.

### Time chart




## Example

### POU header

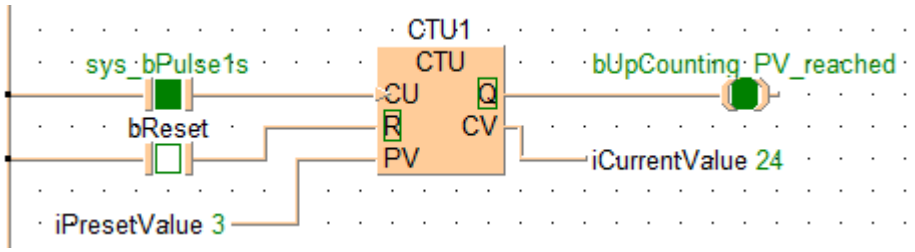
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 1 | VAR | CTU1 | CTU | | |
| 2 | VAR | bReset | BOOL | FALSE | sets CV to zero if TRUE |
| 3 | VAR | iPresetValue | INT | 3 | if CV reaches this value, |
| 4 | VAR | bUpCounting_PV_reached | BOOL | FALSE | is set, if CV reaches PV |
| 5 | VAR | iCurrentValue | INT | 0 | contains the current value |

### POU body

If **bReset** is set (status = TRUE), **iCurrentValue** (CV) will be reset. If a rising edge is detected at **CU**, the value 1 will be added to **iCurrentValue**. If a rising edge is detected at **CU**, this procedure will be repeated until **iCurrentValue** is greater than/equal to **iPresetValue**. Then, **bUpCounting_PV_reached** will be set to TRUE.

### LD body

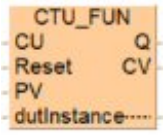

CTU1
CTU
sys_bPulse1s
CU
Q
bUpCounting_PV_reached
bReset
R
CV
PV
iCurrentValue 24
iPresetValue 3

# CTU_FUN

Up counter

This is a user-defined function from a system function block. **CTU_FUN** (count up) allows you to program counting procedures.

Counting up starts at zero until the maximum value 32767 is reached. Each rising edge at count up **CU** increases the value at current value **CV** by 1.




## Parameters

Input

**CU** (BOOL)

count up
the value 1 is added to **CV** for each rising edge at **CU**, except when **Reset** is set

**Reset** (BOOL)

Reset
A rising edge at **Reset** resets the currrent value **CV** to zero and counting stops. The output **Q** is set to FALSE.
The next falling edge at **Reset** restarts counting.

**PV** (INT)

Preset value
if **CV** is equal/greater than **PV**, **Q** is set to TRUE
If no preset value is set or the preset value is zero, the output **Q** is set to TRUE immediately after counting starts.

Input/output

**dutInstance**(CTU_FUN_INSTANCE_DUT)

Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

Output

**Q** (BOOL)

signal output
is set to TRUE if **CV** is greater than/equal to **PV**

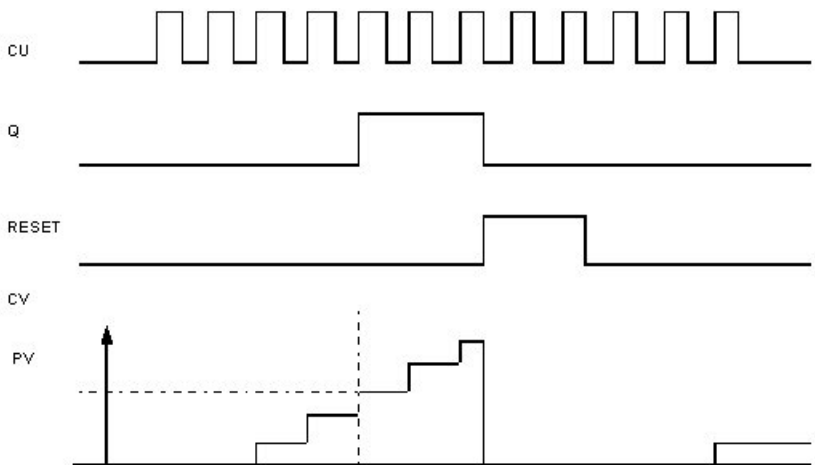
is set to FALSE if a rising edge is detected at **Reset**

**CV** (INT)

Current value

- contains the addition result
- If **CV** reaches the preset value **PV**, the output **Q** is set to TRUE, but counting continues until the maximum limit 32767 is reached.
- The value can be changed during counting operation by writing to the variable from the programming editor.

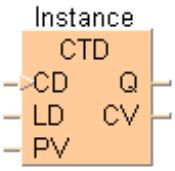
## Time chart



# CTD

Down counter

The function block **CTD** (count down) allows you to program counting procedures.




## Parameters

Input

**CD** (BOOL)

count down
the value 1 is subtracted from the current value **CV** for each rising edge detected at **CD**, except when **LD** is set or **CV** has reached the value zero.

**LD** (BOOL)

Load
with **LD** the counter state is reset to **PV**

**PV** (INT)

Preset value
is the value subjected to subtraction during the first counting procedure

Output

**Q** (BOOL)

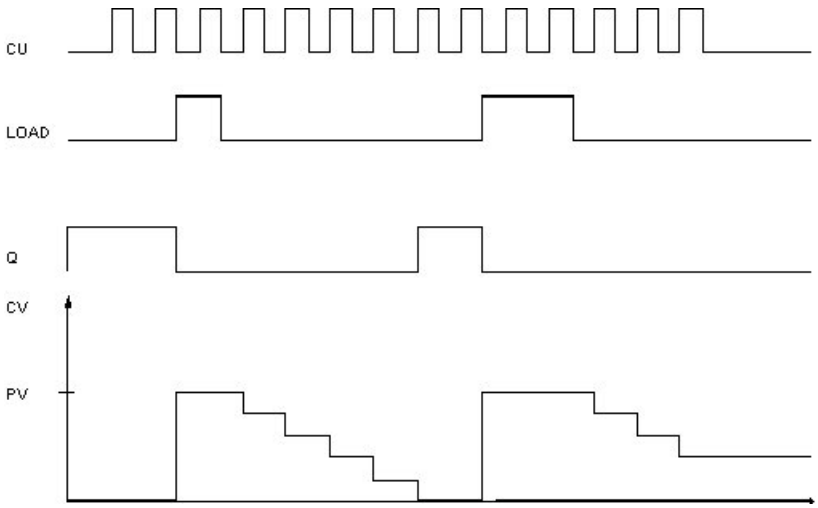
signal output
is set if **CV** = zero

**CV** (INT)

Current value

- contains the current subtraction result (**CV** = current value)
- The value can be changed during counting operation by writing to the variable from the programming editor.

### Time chart




## Example

### POU header

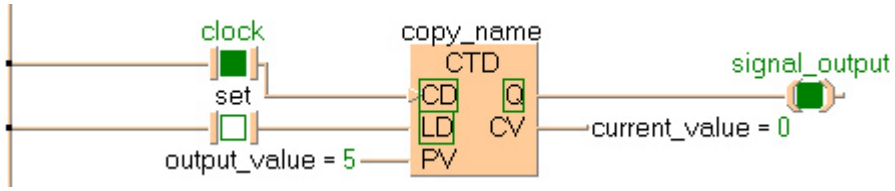
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | copy_name | CTD | | under this identifier a copy of the |
| 1 | VAR | clock | BOOL | FALSE | downward counter input |
| 2 | VAR | set | BOOL | FALSE | set input (set to preset value (PV)) |
| 3 | VAR | output_value | INT | 0 | minuend |
| 4 | VAR | signal_output | BOOL | FALSE | |
| 5 | VAR | current_value | INT | 0 | current counter value |

### POU body

If **set** is set (status = TRUE), the **preset_value** (PV) is loaded in the **current_value** (CV). The value 1 will be subtracted from the **current_value** each time a rising edge is detected at **clock**. This procedure will be repeated until the **current_value** is greater than/equal to zero. Then, **signal_output** will be set.

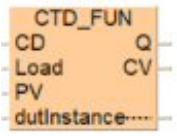
### LD body



# CTD_FUN

Down counter

This is a user-defined function from a system function block. **CTD_FUN** (count down) allows you to program counting procedures.




## Parameters

Input

**CD** (BOOL)

count down
the value 1 is subtracted from the current value **CV** for each rising edge detected at **CD**, except when **Load** is set or **CV** has reached the value zero.

**Load** (BOOL)

with **Load** the counter state is reset to **PV**

**PV** (INT)

Preset value
is the value subjected to subtraction during the first counting procedure

Input/output

**dutInstance**(CTD_FUN_INSTANCE_DUT)

Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

Output

**Q** (BOOL)

signal output
is set if **CV** = zero

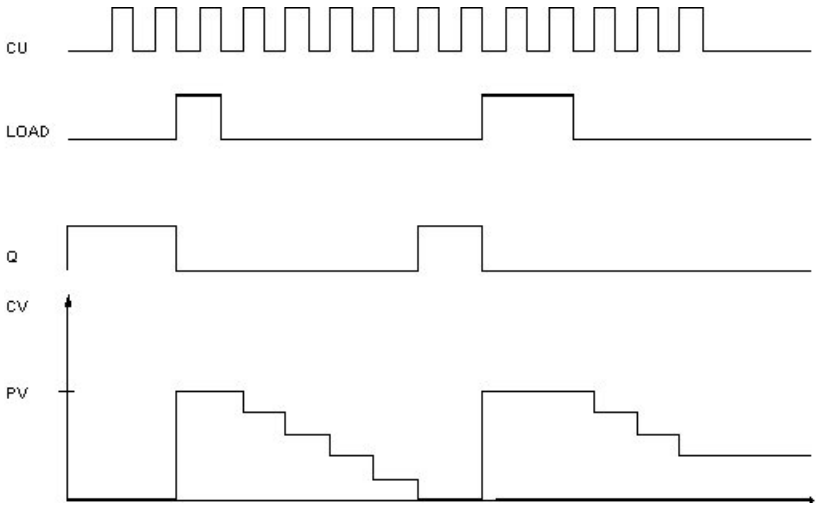
**CV** (INT)

Current value

- contains the current subtraction result (**CV** = current value)

- The value can be changed during counting operation by writing to the variable from the programming editor.

### Time chart



# CTUD

Up/down counter

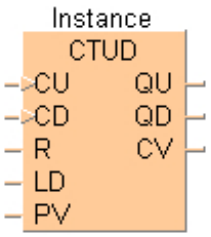
The function block **CTUD** (count up/down) allows you to program counting procedures (up and down).

- Count up (**CU** set to TRUE)

  Counting up starts at zero until the maximum value 32767 is reached. Each rising edge at count up **CU** increases the value at current value **CV** by 1.

- Count down (**CD** set to TRUE)

  Counting down starts at zero until the minimum value -32768 is reached. Each rising edge at count down **CD** decreases the value at current value **CV** by 1.




## Parameters

Input

**CU** (BOOL)

count up

the value 1 is added to the current **CV** for each rising edge detected at **CU**, except when **R** and/or **LD** is/are set.

**CD** (BOOL)

count down

the value 1 is subtracted from the current **CV** for each rising edge detected at **CD**, except when **R** and/or **LD** is/are set

if **CU** and **CD** are simultaneously set to TRUE no counting operation takes place.

**R** (BOOL)

Reset

**CV** is reset to zero for each rising edge at **R** and counting stops. The output **QU**/**QD** is set to FALSE.

The next falling edge at **R** restarts counting.

**LD** (BOOL)

Load

if **LD** is set, **PV** is loaded to **CV** and **QU** is set to TRUE. This, however, does not apply if **R** is set simultaneously. In this case, **LD** will be ignored.

**PV** (INT)

Preset value
defines the preset value which is to be attained with the addition or subtraction
If no preset value is set or the preset value is zero, the output **QU** is set to TRUE immediately after counting starts.

Output

**QU** (BOOL)

signal output - count up
is set to TRUE if **CV** is greater than/equal to **PV**
is set to FALSE if a rising edge is detected at **R**

**QD** (BOOL)

signal output - count down
is set to TRUE if **CV** = zero
is set to FALSE if a rising edge is detected at **R**

**CV** (INT)

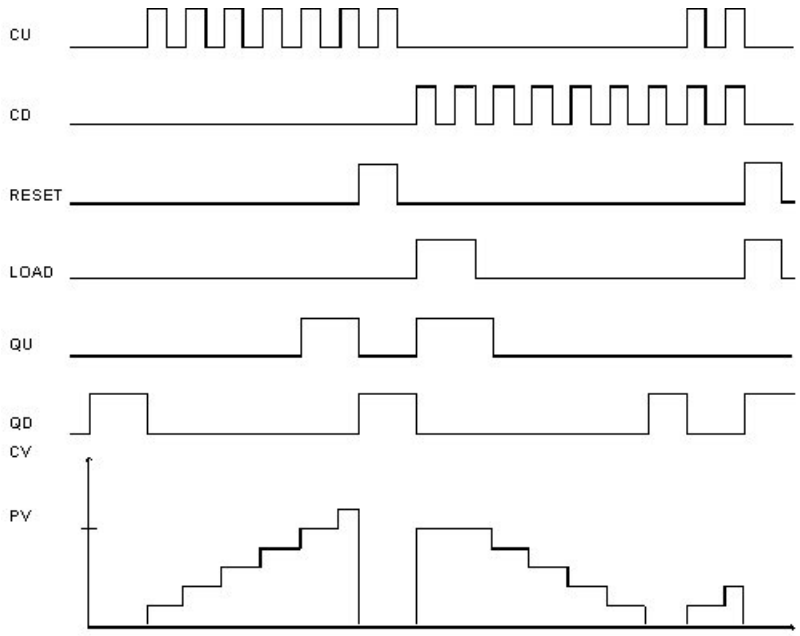
Current value

- is the addition/subtraction result
- counting up:

  If **CV** reaches the preset value **PV**, the output **QU** is set to TRUE, but counting continues until the maximum limit 32767 is reached.
- counting down:

  If **CV** reaches zero, the output **QD** is set to TRUE, but counting continues until the minimum limit -32768 is reached.
- The value can be changed during counting operation by writing to the variable from the programming editor.

### Time chart




## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 1 | VAR | CTUD1 | CTUD | | |
| 2 | VAR | bCountUp | BOOL | FALSE | increments CV if TRUE |
| 3 | VAR | bCountDown | BOOL | FALSE | decrements CV if TRUE |
| 4 | VAR | bReset | BOOL | FALSE | sets CV to zero if TRUE |
| 5 | VAR | bLoad | BOOL | FALSE | sets CV to PV if TRUE |
| 6 | VAR | iPresetValue | INT | 3 | if CV reaches this value, |
| 7 | VAR | bUpCounting_PV_reached | BOOL | FALSE | is set, if CV reaches PV |
| 8 | VAR | bDownCounting_zero_reached | BOOL | FALSE | is set, if CV reaches zero |
| 9 | VAR | iCurrentValue | INT | 0 | contains the current value |

### POU body

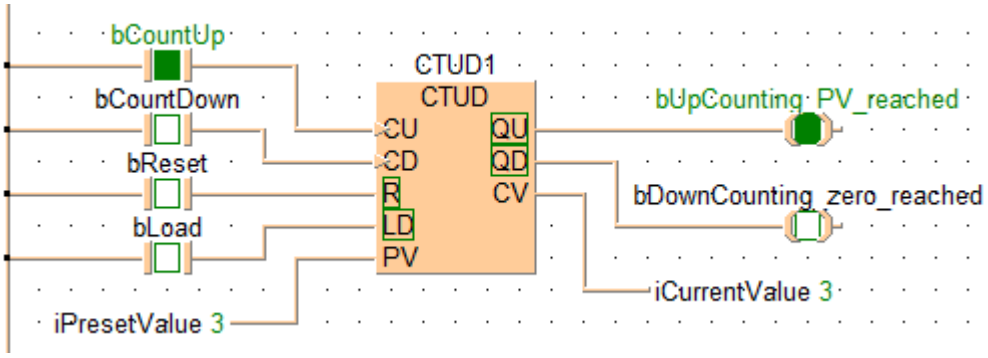
Count up:

If **bReset** is set, the **iCurrentValue** (CV) will be reset to zero. If **bCountUp** is set, the value 1 is added to the **iCurrentValue**. This procedure is repeated for each rising edge detected at **bCountUp** until the **iCurrentValue** is greater than/equal to the **iPresetValue**. Then **bUpCounting_PV_reached** is set. The procedure is not conducted, if **bReset** and/or **bLoad** is/are set.

Count down:

If **bReset** is set (status = TRUE), the **iPresetValue** (PV = preset value) will be loaded into **iCurrentValue**. If **bCountDown** is set, the value 1 is subtracted from **iPresetValue**. This procedure is repeated at each rising edge until the iCurrentValue is smaller than/equal to zero. Then, **bDownCounting_zero_reached** is set to TRUE. The procedure will not be conducted, if **bReset** and/or **bLoad** is/are set. If **CU** and **CD** are set at the same time, no counting operation takes place.

## LD body



# CTUD_FUN

Up/down counter

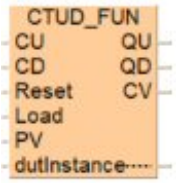
This is a user-defined function from a system function block. **CTUD_FUN** (count up/down) allows you to program counting procedures (up and down).

- Count up (**CU** set to TRUE)

  Counting up starts at zero until the maximum value 32767 is reached. Each rising edge at count up **CU** increases the value at current value **CV** by 1.

- Count down (**CD** set to TRUE)

  Counting down starts at zero until the minimum value -32768 is reached. Each rising edge at count down **CD** decreases the value at current value **CV** by 1.




## Parameters

Input

**CU** (BOOL)

count up
the value 1 is added to the current **CV** for each rising edge detected at **CU**, except when **Reset** and/or **Load** is/are set.

**CD** (BOOL)

count down
the value 1 is subtracted from the current **CV** for each rising edge detected at **CD**, except when **Reset** and/or **Load** is/are set
if **CU** and **CD** are simultaneously set to TRUE no counting operation takes place.

**Reset** (BOOL)

Reset
**CV** is reset to zero for each rising edge at **Reset** and counting stops. The output **QU**/**QD** is set to FALSE.
The next falling edge at **Reset** restarts counting.

**Load** (BOOL)

if **Load** is set, **PV** is loaded to **CV** and **QU** is set to TRUE. This, however, does not apply if **Reset** is set simultaneously. In this case, **Load** will be ignored.

**PV** (INT)

Preset value

defines the preset value which is to be attained with the addition or subtraction

If no preset value is set or the preset value is zero, the output **QU** is set to TRUE immediately after counting starts.

Input/output

**dutInstance**(CTUD_FUN_INSTANCE_DUT)

Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

Output

**QU** (BOOL)

signal output - count up

is set to TRUE if **CV** is greater than/equal to **PV**

is set to FALSE if a rising edge is detected at **Reset**

**QD** (BOOL)

signal output - count down

is set to TRUE if **CV** = zero

is set to FALSE if a rising edge is detected at **Reset**

**CV** (INT)

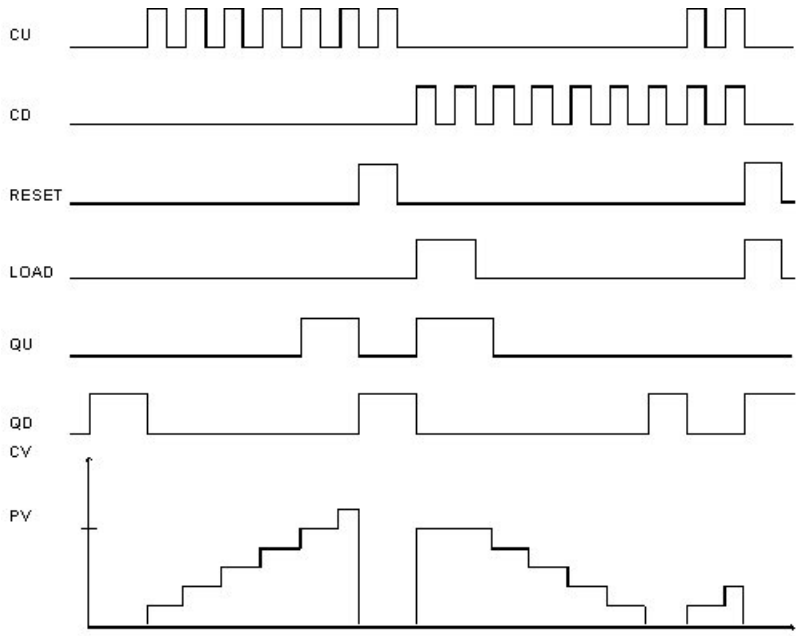
Current value

- is the addition/subtraction result
- counting up:

  If **CV** reaches the preset value **PV**, the output **QU** is set to TRUE, but counting continues until the maximum limit 32767 is reached.
- counting down:

  If **CV** reaches zero, the output **QD** is set to TRUE, but counting continues until the minimum limit -32768 is reached.
- The value can be changed during counting operation by writing to the variable from the programming editor.

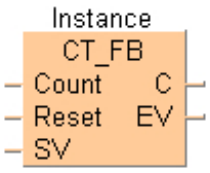
**Time chart**



# CT_FB

Down counter

Counters realized with the **CT_FB** function block are down counters. The count area **SV** (set value) is 1 to 32767.




## Parameters

Input

**Count** (BOOL)

count contact (down)

each time a rising edge is detected at **Count**, the value 1 is subtracted from the elapsed value **EV** until the value 0 is reached

**Reset** (BOOL)

reset contact

each time a rising edge is detected at **Reset**, the value 0 is assigned to **EV** and the signal output **C** is reset; each time a falling edge is detected at **Reset**, the value at **SV** is assigned to **EV**

**SV** (INT)

Set value

value of **EV** after a reset procedure

Output

**C** (BOOL)

signal output

is set when **EV** becomes 0

**EV** (INT)

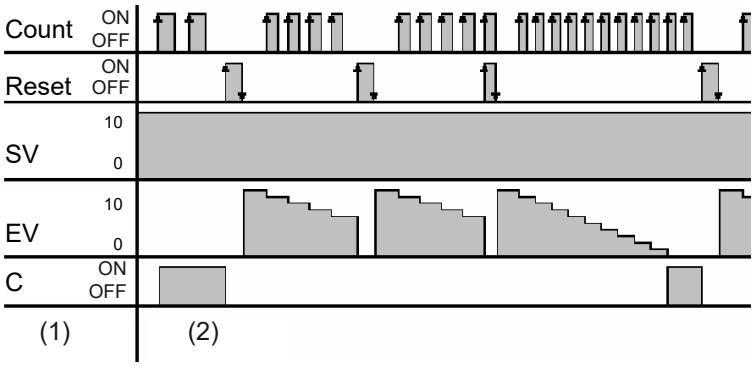
Elapsed value

- current counter value
- The value can be changed during counting operation by writing to the variable from the programming editor.

### Remarks

- In order to work correctly, the **CT_FB** function block needs to be reset each time before it is used.
- The number of available counters is limited and depends on the settings in the system registers 5 and 6. The compiler assigns a NUM* address to every counter instance. The addresses are assigned counting downwards, starting at the highest possible address.
- The basic function **CT** (down counter) uses the same NUM* address area (**Num** input). In order to avoid errors (address conflicts), the **CT** function and the **CT_FB** function block should not be used together in a project.

Time chart




(1) download PROG mode
(2) RUN mode

## Example

### POU header

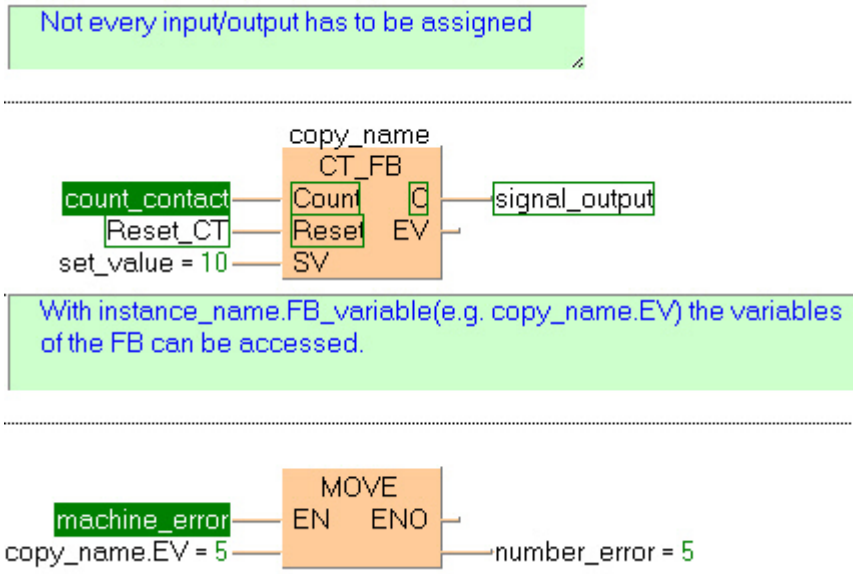
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | copy_name | CT_FB | |
| 1 | VAR | set_value | INT | 10 |
| 2 | VAR | signal_output | BOOL | FALSE |
| 3 | VAR | count_contact | BOOL | FALSE |
| 4 | VAR | Reset_CT | BOOL | FALSE |
| 5 | VAR | machine_error | BOOL | FALSE |
| 6 | VAR | number_error | INT | 0 |

### POU body

This example uses variables. You may also use constants for the input variables. Each rising edge detected at **count_contact** the value 1 is subtracted from the elapsed value **EV**. **Signal_output** is set to TRUE if the elapsed value **EV** becomes zero.

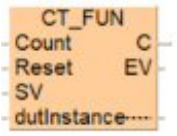
### LD body


Not every input/output has to be assigned
copy_name
CT_FB
count_contact
Reset_CT
set_value = 10
Count
Reset
SV
C
EV
signal_output
With instance_name.FB_variable(e.g. copy_name.EV) the variables of the FB can be accessed.
MOVE
machine_error
copy_name.EV = 5
EN
ENO
number_error = 5

# CT_FUN

Down counter

This is a user-defined function from a system function block. Counters realized with the **CT_FUN** function are down counters. The count area **SV** (set value) is 1 to 32767.




## Parameters

Input

**Count** (BOOL)

count contact (down)
each time a rising edge is detected at **Count**, the value 1 is subtracted from the elapsed value **EV** until the value 0 is reached

**Reset** (BOOL)

reset contact
each time a rising edge is detected at **Reset**, the value 0 is assigned to **EV** and the signal output **C** is reset; each time a falling edge is detected at **Reset**, the value at **SV** is assigned to **EV**

**SV** (INT)

Set value
value of **EV** after a reset procedure

Input/output

**dutInstance**(CT_FUN_INSTANCE_DUT)

Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

Output

**C** (BOOL)

signal output
is set when **EV** becomes 0

**EV** (INT)

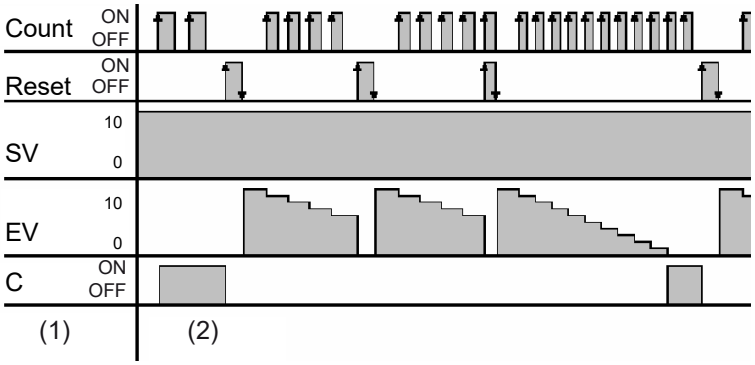
Elapsed value

- current counter value
- The value can be changed during counting operation by writing to the variable from the programming editor.

## Remarks

- In order to work correctly, the **CT_FUN** function needs to be reset each time before it is used.
- The number of available counters is limited and depends on the settings in the system registers 5 and 6. The compiler assigns a NUM* address to every counter instance. The addresses are assigned counting downwards, starting at the highest possible address.
- The basic function **CT** (down counter) uses the same NUM* address area (**Num** input). In order to avoid errors (address conflicts), the **CT** function and the **CT_FUN** function should not be used together in a project.

Time chart




(1) download PROG mode
(2) RUN mode

## 14.9 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

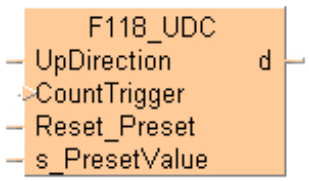
Advantages of IEC instructions

Advantages of FP instructions

# F118_UDC

UP/DOWN counter

DOWN counting if the trigger **UpDirection** is in the OFF state. UP counting if the trigger **UpDirection** is in the ON state.




## Parameters

Input

**UpDirection** (BOOL)

sets counter to count up (ON) or down (0FF)

**CountTrigger** (BOOL)

starts counter

**Reset_Preset** (BOOL)

resets counter

**s_PresetValue** (WORD, INT, UINT)

16-bit area or equivalent constant for counter preset value

Output

**d** (WORD, INT, UINT)

16-bit area for counter elapsed value

## Remarks

- The area for the elapsed value **d** becomes 0 when the rising edge of the trigger is detected (OFF → ON). The value in **s_PresetValue** is transferred to **d** when the falling edge of the trigger is detected (ON → OFF).
- The variables **s_PresetValue** and **d** have to be of the same data type.

## Example

### POU header

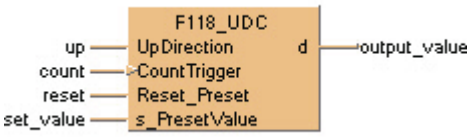
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | up | BOOL | FALSE | declares, if the counter |
| 1 | VAR | count | BOOL | FALSE | at a rising edge on count |
| 2 | VAR | reset | BOOL | FALSE | resets the counter to |
| 3 | VAR | set_value | INT | 0 | the starting value |
| 4 | VAR | output_value | INT | 0 | the actual value |

### POU body

A rising edge at the input **Count_Trigger** activates the counter. The boolean variable at the input **UpDirection** sets the direction of the counter (TRUE = up, FALSE =down). TRUE at the input **Reset_Preset** resets the counter to the starting value.

### LD body



# 15 Data table instructions

## 15.1 FP instructions

Tip

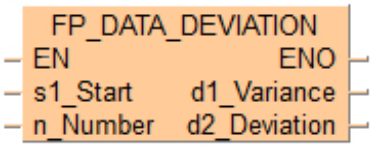
Advantages of FP instructions

### 15.1.1 Data table analysis FP instructions

# FP_DATA_DEVIATION

Calculate deviation from data table

This FP instruction calculates the deviation from the values in a data table.




## Parameters

Input

**s1_Start** (WORD, INT, UINT)

Beginning of data table

**n_Number** (WORD, INT, UINT)

Number of values

Output

**d1_Variance** (REAL)

Variance, deviation$^2$

**d2_Deviation** (REAL)

Standard deviation, square root of variance

## Example

## POU header

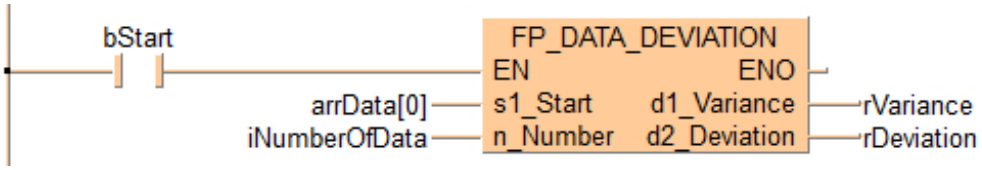
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | arrData | ARRAY [0..31] OF INT | [0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,1] |
| 1 | VAR | rDeviation | REAL | 0.0 |
| 2 | VAR | rVariance | REAL | 0.0 |
| 3 | VAR | iNumberOfData | INT | 8 |
| 4 | VAR | bStart | BOOL | FALSE |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

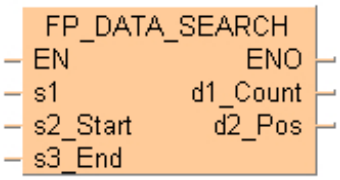
### LD body



# FP_DATA_SEARCH

Data search

This FP instruction searches for the value specified at **s1** in the range specified by **s2_Start** and **s3_End** if the trigger **EN** is TRUE.




## Parameters

Input

**s1** (INT, DINT, UINT, UDINT, REAL, LREAL)

Data to be searched for

**s2_Start** (INT, DINT, UINT, UDINT, REAL, LREAL)

Beginning of search range

**s3_End** (INT, DINT, UINT, UDINT, REAL, LREAL)

End of search range

Output

**d1_Count** (DINT)

Number of matches found

**d2_Pos** (DINT)

Relative position of first match (first position is 0)

## Remarks

- Make sure that **s2_Start**≤**s3_End**
- The variables **s1**, **s2_Start** and **s3_End** have to be of the same data type.
- The maximum number of data that can be searched is 30000.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

- if **s2_Start>s3_End**
- if **s2_Start** and **s3_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s2_Start>s3_End**
- if **s2_Start** and **s3_End** belong to different data areas.

## Example

### POU header

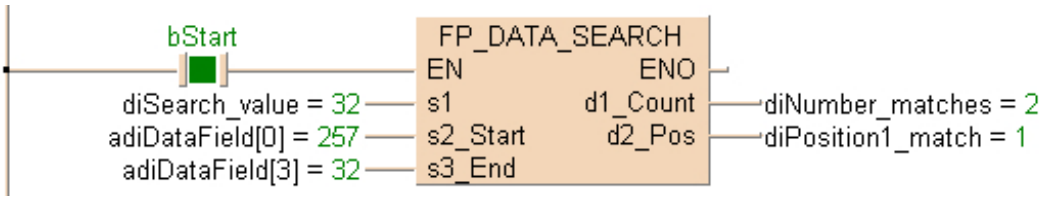
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the fuction |
| 1 | VAR | diSearch_value | DINT | 16#20 | specifies the value to |
| 2 | VAR | adiDataField | ARRAY [0..3] OF DINT | [16#101,16#20,... | 2 matches for 16#20 |
| 3 | VAR | diNumber_matches | DINT | 0 | |
| 4 | VAR | diPosition1_match | DINT | 0 | |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

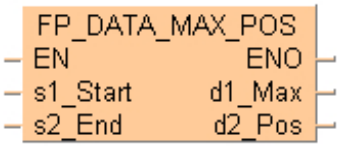
### LD body



# FP_DATA_MAX_POS

Search maximum value in data table

This FP instruction determines the maximum value and its position in a data table.




## Parameters

Input

**s1_Start** (INT, DINT, UINT, UDINT, REAL, LREAL)

Beginning of data table

**s2_End** (INT, DINT, UINT, UDINT, REAL, LREAL)

End of data table

Output

**d1_Max** (INT, DINT, UINT, UDINT, REAL, LREAL)

Maximum value

**d2_Pos** (DINT)

Position of maximum value

## Remarks

- Input **s1_Start** specifies the beginning of the data table, and **s2_End** specifies the end. The maximum value is returned at output **d1_Max** and its position at output **d2_Pos**.
- The position **d2_Pos** is the position of the first occurrence of the maximum value relative to the beginning of the data table.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

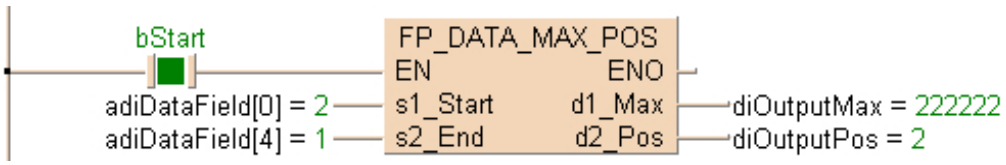
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | adiDataField | ARRAY [0..4] OF DINT | [2,3,222222,-... | |
| 2 | VAR | diOutputMax | DINT | 0 | the maximum value of |
| 3 | VAR | diOutputPos | DINT | 0 | the position of the |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

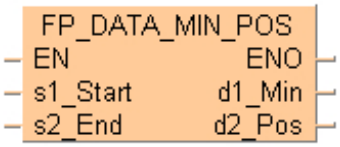
### LD body



# FP_DATA_MIN_POS

Search minimum value in data table

This FP instruction determines the minimum value and its position in a data table.




## Parameters

Input

**s1_Start** (INT, DINT, UINT, UDINT, REAL, LREAL)

Beginning of data table

**s2_End** (INT, DINT, UINT, UDINT, REAL, LREAL)

End of data table

Output

**d1_Min** (INT, DINT, UINT, UDINT, REAL, LREAL)

Minimum value

**d2_Pos** (DINT)

Position of minimum value

## Remarks

- Input **s1_Start** specifies the beginning of the data table, and **s2_End** specifies the end. The minimum value is returned at output **d1_Min** and its position at output **d2_Pos**.
- The position **d2_Pos** is the position of the first occurrence of the minimum value relative to the beginning of the data table.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

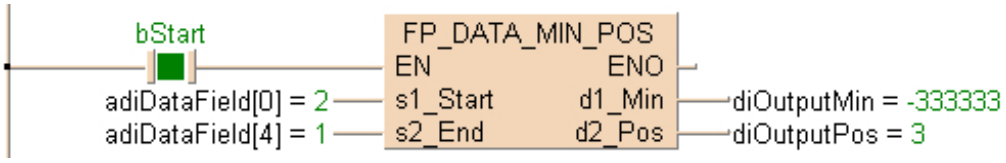
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | adiDataField | ARRAY [0..4] OF DINT | [2,3,2222... | |
| 2 | VAR | diOutputMin | DINT | 0 | the maximum value of |
| 3 | VAR | diOutputPos | DINT | 0 | the position of the |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

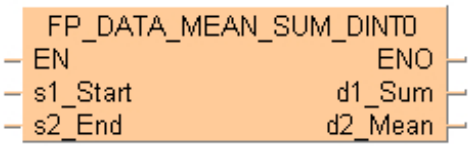
### LD body



# FP_DATA_MEAN_SUM_DINT0

Calculate total and mean numbers in data table of DINT numbers

This FP instruction calculates the sum and the arithmetic mean of positive and negative DINT numbers in the data table specified by **s1_Start** and **s2_End**. The sum of all elements in the data table is returned at output **d1_Sum** and the arithmetic mean of all elements in the data table is returned at output **d2_Mean**. The arithmetic mean is rounded off if it is not a whole number.




## Parameters

Input

**s1_Start** (DINT)

Beginning of data table

**s2_End** (DINT)

End of data table

Output

**d1_Sum** (ARRAY[0..1] OF DINT)

Sum of all elements in data table area specified

**d2_Mean** (DINT)

Mean of all elements in data table area specified

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start**>**s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start**>**s2_End**

- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

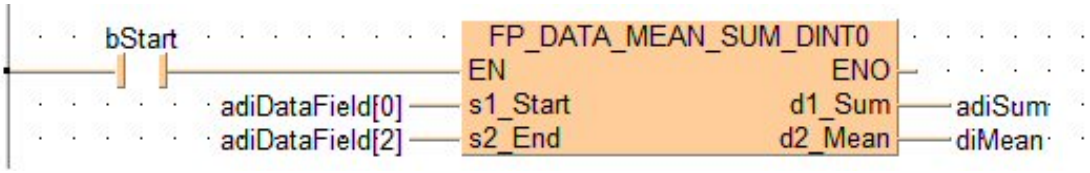
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 1 | VAR | bStart | BOOL | FALSE | activates the function |
| 2 | VAR | adiDataField | ARRAY [0..2] OF DINT | [3,-1,4] | |
| 3 | VAR | adiSum | ARRAY [0..1] OF DINT | [2(0)] | the sum of all elements of data_array; here: 6 |
| 4 | VAR | diMean | DINT | 0 | the arithmetic mean of all elements of data_array; here: 2 |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

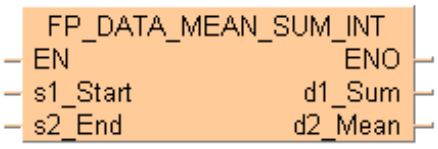
### LD body



# FP_DATA_MEAN_SUM_INT

Calculate total and mean numbers in data table of INT numbers

This FP instruction calculates the sum and the arithmetic mean of positive and negative INT numbers in the data table specified by **s1_Start** and **s2_End**. The sum of all elements in the data table is returned at output **d1_Sum** and the arithmetic mean of all elements in the data table is returned at output **d2_Mean**. The arithmetic mean is rounded off if it is not a whole number.




## Parameters

Input

**s1_Start** (INT)

Beginning of data table

**s2_End** (INT)

End of data table

Output

**d1_Sum** (INT)

Sum of all elements in data table area specified

**d2_Mean** (INT)

Mean of all elements in data table area specified

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start**>**s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start**>**s2_End**

- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

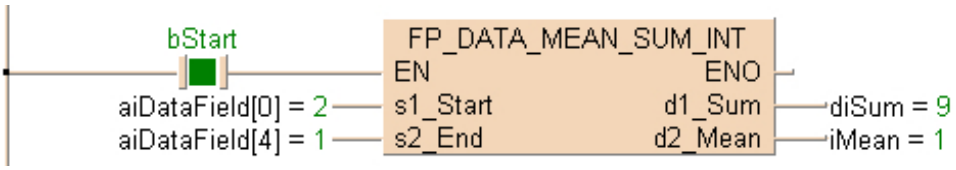
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | aiDataField | ARRAY [0..4] OF INT | [2,3,6,-3,1] | |
| 2 | VAR | diSum | DINT | 0 | the sum of all elements of |
| 3 | VAR | iMean | INT | 0 | the arithmetic mean of all |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

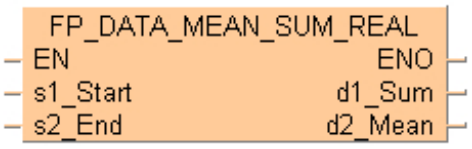
### LD body



# FP_DATA_MEAN_SUM_REAL

Calculate total and mean numbers in data table of REAL numbers

This FP instruction calculates the sum and the arithmetic mean of positive and negative REAL numbers in the data table specified by **s1_Start** and **s2_End**. The sum of all elements in the data table is returned at output **d1_Sum** and the arithmetic mean of all elements in the data table is returned at output **d2_Mean**. The arithmetic mean is rounded off if it is not a whole number.




## Parameters

Input

**s1_Start** (ANY_REAL)

Beginning of data table

**s2_End** (ANY_REAL)

End of data table

Output

**d1_Sum** (ANY_REAL)

Sum of all elements in data table area specified

**d2_Mean** (ANY_REAL)

Mean of all elements in data table area specified

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start**>**s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start**>**s2_End**

- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

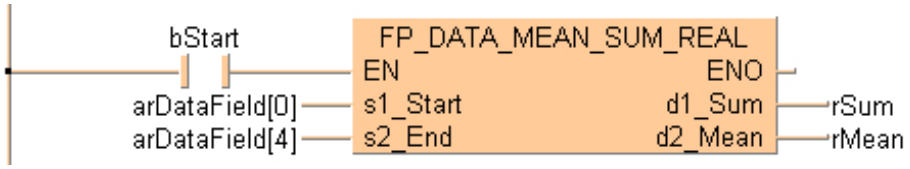
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | arDataField | ARRAY [0..4] OF REAL | [2.102,3.33,6.61,2.02,-4.14] | |
| 2 | VAR | rSum | REAL | 0 | the sum of all elements of |
| 3 | VAR | rMean | REAL | 0 | the arithmetic mean of all |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

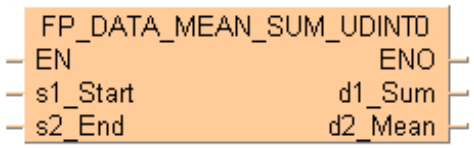
### LD body



# FP_DATA_MEAN_SUM_UDINT0

Calculate total and mean numbers in data table of UDINT numbers

This FP instruction calculates the sum and the arithmetic mean of UDINTnumbers in the data table specified by **s1_Start** and **s2_End**. The sum of all elements in the data table is returned at output **d1_Sum** and the arithmetic mean of all elements in the data table is returned at output **d2_Mean**. The arithmetic mean is rounded off if it is not a whole number.




## Parameters

Input

**s1_Start** (UDINT)

Beginning of data table

**s2_End** (UDINT)

End of data table

Output

**d1_Sum** (ARRAY[0..1] OF UDINT)

Sum of all elements in data table area specified

**d2_Mean** (UDINT)

Mean of all elements in data table area specified

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start** >**s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start** >**s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

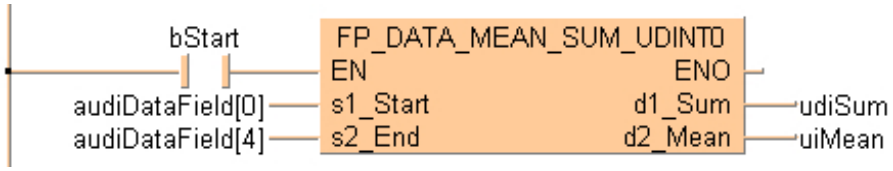
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | audiDataField | ARRAY [0..4] OF UDINT | [31111,42440,5... | |
| 2 | VAR | udiSum | UDINT | 0 | the sum of all elements of |
| 3 | VAR | uiMean | UINT | 0 | the arithmetic mean of all |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

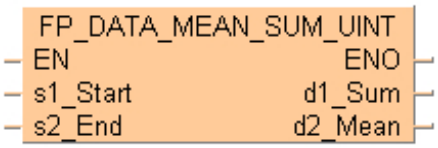
### LD body



# FP_DATA_MEAN_SUM_UINT

Calculate total and mean numbers in data table of UINT numbers

This FP instruction calculates the sum and the arithmetic mean of UINT numbers in the data table specified by **s1_Start** and **s2_End**. The sum of all elements in the data table is returned at output **d1_Sum** and the arithmetic mean of all elements in the data table is returned at output **d2_Mean**. The arithmetic mean is rounded off if it is not a whole number.




## Parameters

Input

**s1_Start** (UINT)

Beginning of data table

**s2_End** (UINT)

End of data table

Output

**d1_Sum** (UDINT)

Sum of all elements in data table area specified

**d2_Mean** (UINT)

Mean of all elements in data table area specified

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start** >**s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Start** >**s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

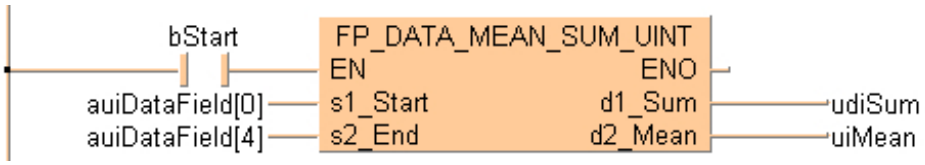
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | auiDataField | ARRAY [0..4] OF UINT | [2,4,6,8,5] | |
| 2 | VAR | udiSum | UDINT | 0 | the sum of all elements of |
| 3 | VAR | uiMean | UINT | 0 | the arithmetic mean of all |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

### LD body

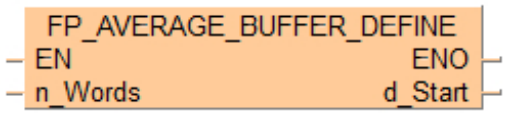


### 15.1.2 Data table manipulation FP instructions

# FP_AVERAGE_BUFFER_DEFINE

Define buffer area for moving average and total values

This FP instruction specifies a special buffer to compute the average value and the total of all contained values. The starting address is set by **d_Start** and the size by **n_Words**. The buffer has a particular structure, which is shown in the programming example, where the number of array elements of the last element **iArrData** corresponds to the size defined by **n_words**. Use the instruction **FP_AVERAGE_BUFFER_WRITE** for writing into the buffer.




## Parameters

Input

**n_Words** (WORD, INT, UINT)

Specifies the memory size of the buffer
Values: 1–30000

Output

**d_Start** (WORD, INT, UINT)

Starting data area of the buffer
We recommend using a user-defined DUT as shown in the programming example.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.

## Example

### DUT

DUT_FP_AVERAG...UFFER_INT_10

| | Identifier | Type | Initial |
|---|---|---|---|
| 0 | uiBufferSize | UINT | 0 |
| 1 | uiNumberOfStoredData | UINT | 0 |
| 2 | diTotalValueOfStoredData | DINT | 0 |
| 3 | rMovingAverage | REAL | 0.0 |
| 4 | uiWritePointer | UINT | 0 |
| 5 | iArrData | ARRAY [0..9] OF INT | [10(0)] |

Elements of the DUT (identifiers):

**uiBufferSize**

Size of buffer

**uiNumberOfStoredData**

Number of stored data

**diTotalValueOfStoredData**

Total sum of stored data

**rMovingAverage**

Moving average

**uiWritePointer**

Position of writing pointer

**iArrData**

Contains the values of the buffer. The number of integer values (10 in this example) must be equal to the size of the buffer defined with **iNumberOfIntegerValues**.

### POU header

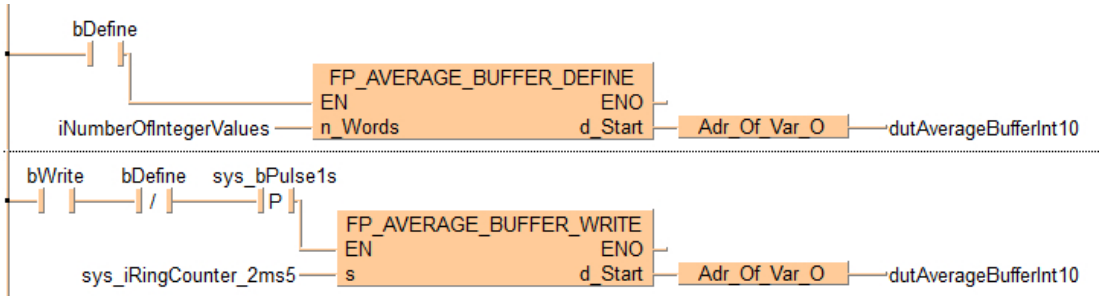
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bDefine | BOOL | FALSE | |
| 1 | VAR | bWrite | BOOL | FALSE | |
| 2 | VAR | iCounter | INT | 0 | |
| 3 | VAR_CONSTANT | iNumberOfIntegerValues | INT | 10 | number of integers values must be equal to average buffer size |
| 4 | VAR | dutAverageBufferInt10 | DUT_FP_AVERAGE_BUFFER_INT_10 | | Average Buffer for integer numbers of size 10 |

## POU body

When the variable **bDefine** is set to TRUE, the function is carried out.

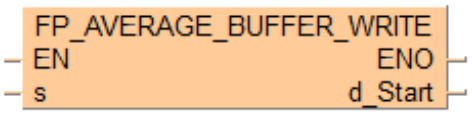
## LD body



# FP_AVERAGE_BUFFER_WRITE

Write to buffer for moving average and total values

This FP instruction writes the data specified by **s** into the buffer for moving average and total values specified by **d_Start**.




## Parameters

Input

**s** (INT), (UINT)

Data area or equivalent constant for storing data to write in the buffer

Output

**d_Start** (WORD, INT, UINT)

Starting data area of the buffer

## Remarks

If this instruction is executed when the writing pointer is indicating the final address in the buffer (**n_Words** defined by **FP_AVERAGE_BUFFER_DEFINE**), the writing pointer will be set to 0.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the size (**n_Words**) of the buffer specified by **d_Start** is **n_Words** = 0, or when **n_Words** > 30000.
- if the number of stored data items of the buffer = 0.
- if the number of stored data items of the buffer > buffer size (**n_Words**).
- if the writing pointer > buffer size (**n_Words**).
- if the writing pointer is 30000 (16#7530) or higher after the data has been written.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the size (**n_Words**) of the buffer specified by **d_Start** is **n_Words** = 0, or when **n_Words** > 30000.
- if the number of stored data items of the buffer = 0.

- if the number of stored data items of the buffer > buffer size (**n_Words**).
- if the writing pointer > buffer size (**n_Words**).
- if the writing pointer is 30000 (16#7530) or higher after the data has been written.

## Example

Please refer to the example of FP_AVERAGE_BUFFER_DEFINE.

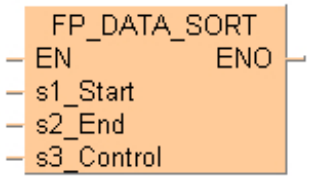
# FP_DATA_SORT

Sort data in data table

This FP instruction sorts values (with +/- sign) in a data table in ascending or descending order.

Note

This instruction sorts 1-dimensional arrays only. It does not work with 2-dimensional or 3-dimensional arrays.




## Parameters

Input

**s1_Start** (INT, DINT, UINT, UDINT, REAL, LREAL)

Starting area of data table to be sorted

**s2_End** (INT, DINT, UINT, UDINT, REAL, LREAL)

Ending area of data table to be sorted

**s3_Control** (INT, DINT, UINT, UDINT, REAL, LREAL)

Specifies sorting order:

- 0 = ascending
- 1 = descending

## Remarks

- The data are sorted via bubble sort in the order specified according to the value entered at input **s3_Control**. Since the number of word comparisons increases in proportion to the square of the number of words, the sorting process can take some time when there is a large number of words. When the address of the variable at input **s1_Start** = **s2_End**, no sorting takes place.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

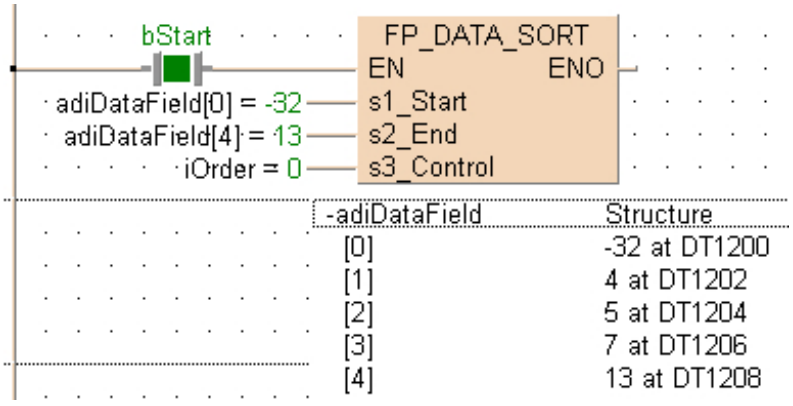
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | adiDataField | ARRAY [0..4] OF DINT | [4,7,-32,13,5] | result after a 0->1 leading |
| 2 | VAR | iOrder | INT | 0 | which way to sort: |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

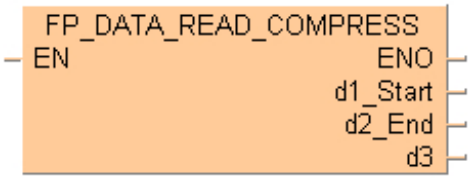
### LD body



# FP_DATA_READ_COMPRESS

Shift out and compress data

This FP instruction shifts out non-zero data stored at the highest address of the table to the specified area and compresses the data in the table to the higher address.




## Parameters

Output

**d1_Start** (WORD, INT, UINT)

Starting (lowest) address of data to be compressed

**d2_End** (WORD, INT, UINT)

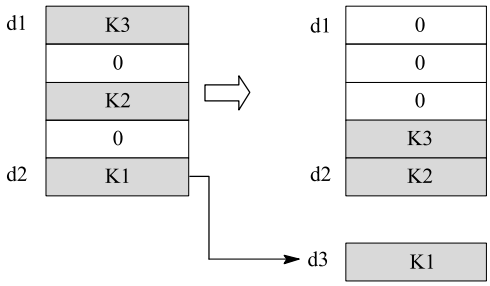
Final (highest) address of data to be compressed, data at **d2_End** is shifted out

**d3** (WORD, INT, UINT)

Receives data shifted out from **d2_End**

## Remarks

- The data in the table specified by **d1_Start** and **d2_End** is rearranged as follows:
  - Contents of **d2_End** (highest address) are shifted out to the area specified by **d3**.
  - Non-zero data is shifted (compressed) in sequential order, in the direction of the higher address in the specified range.




- Starting area **d1_Start** and ending area **d2_End** should be the same type of operand.
- **d1_Start** must be ≤**d2_End**

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

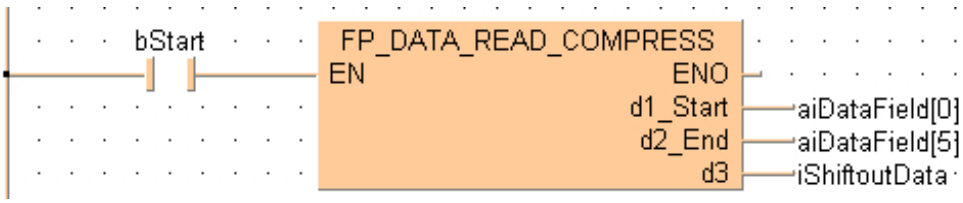
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | aiDataField | ARRAY [0..5] OF INT | [555,444,0,11,0,10] |
| 2 | VAR | iShiftoutData | INT | 10 |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

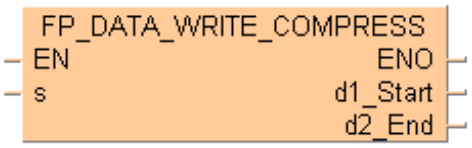
### LD body



# FP_DATA_WRITE_COMPRESS

Shift in and compress data

This FP instruction shifts in data to the smallest address of the specified data table and compresses the data in the table toward the higher address.




## Parameters

Input

**s** (WORD, INT, UINT)

Data to be shifted in

Output

**d1_Start** (WORD, INT, UINT)

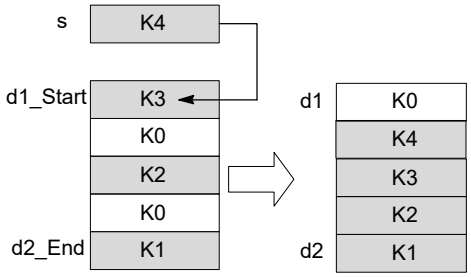
Starting address of area that is compressed into which data from **s** is shifted

**d2_End** (WORD, INT, UINT)

Ending address of area where data is compressed

## Remarks

- The data in the table specified by **d1_Start** and **d2_End** is rearranged as follows:
  - Data specified by **s** is shifted in to the area specified by **d1_Start** (starting address).
  - Non-zero data is shifted (compressed) in sequential order, in the direction of the higher address in the specified range.




- Starting area **d1_Start** and ending area **d2_End** should be the same type of operand.
- **d1_Start** must be ≤**d2_End**
- If the content of **s** is "0", only a compressed shift is carried out.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

### Example

### POU header

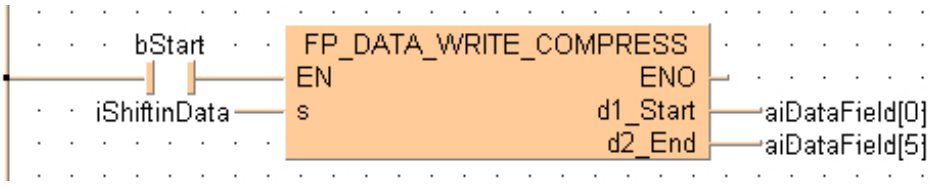
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | aiDataField | ARRAY [0..5] OF INT | [555,444,0,11,0,10] |
| 2 | VAR | iShiftinData | INT | 32 |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

### LD body




## 15.1.2.6 Introduction to the FIFO buffer

The FIFO buffer is a first-in-first-out buffer area realized as a ring buffer. Data is stored in the order in which it is written to the buffer, and then read out in the order stored, starting from the first data item stored. It is convenient for buffering objects in sequential order.

### Usage procedure

- The area to be used is defined as the FIFO buffer using **FP_FIFO_DEFINE**. (This should be done only once, before reading or writing is done.)
- Data should be written to the buffer using **FP_FIFO_WRITE**, and read out of the buffer using **FP_FIFO_READ**.

### Writing data

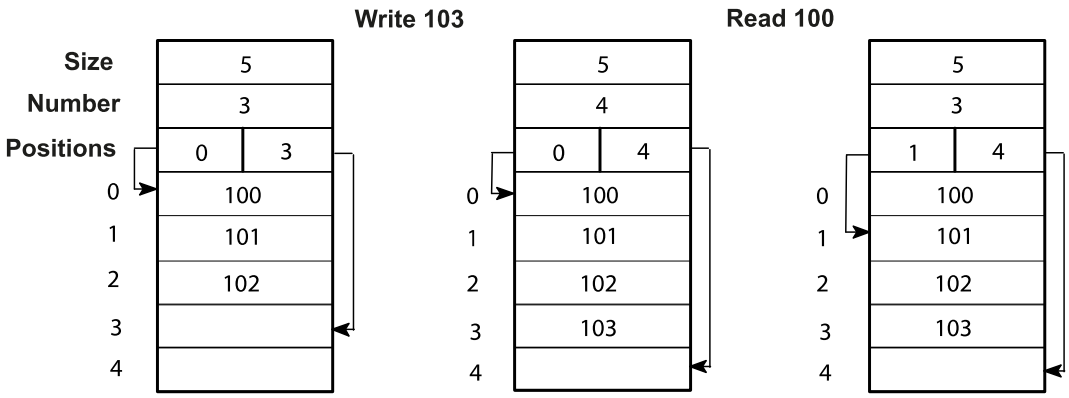
- When data is written, the data items are stored in sequential order, starting from the first data storage area. The writing pointer indicates the next area to which data is to be written. The number of words stored increases by 1.
- If the data storage area becomes full, i.e. the number of words stored is equal to n-1, further data writing is inhibited.

### Reading data

- When data is read, data is transferred starting from the first data item stored. The reading pointer indicates the next area from which data is to be read. The number of words stored decreases by 1.
- An error occurs if an attempt is made to read data when the data storage area is empty, the number of words stored is equal to the memory size of the FIFO buffer or is equal to zero.

### Data storage area

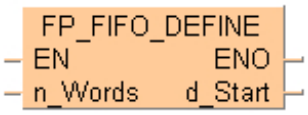
- If data is written while the FIFO buffer is in the status shown below, the data will be stored in the area indicated by 3. The writing pointer moves to 4, i.e. the next data item will be written to 4. If data is read, it will be read from the area indicated by 0. The reading pointer then moves to 1, i.e. the next data item will be read from 1.


Write 103
Read 100
Size
Number
Positions
5
3
0
3
100
101
102
4
0
4
103
1
0
1
2
3
4

# FP_FIFO_DEFINE

Define the FIFO buffer area

This FP instruction specifies the starting area **d_Start** for the FIFO (First-In-First-Out) buffer and the memory size **n_Words** of the FIFO buffer.




## Parameters

Input

**n_Words** (WORD, INT, UINT)

Specifies the memory size of the buffer
Values: 1–4096

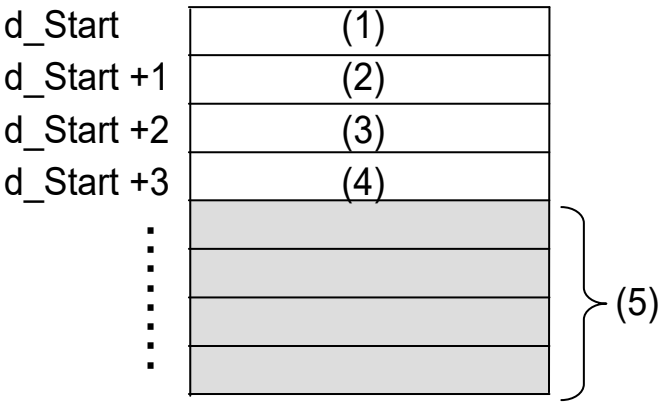
Output

**d_Start** (WORD, INT, UINT)

Starting data area of the buffer

## Remarks

Format of data buffer




(1) Buffer size
(2) Number of stored data
(3) Reading pointer
(4) Writing pointer
(5) Data buffer area (not cleared)

### Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.

## Example

### POU header

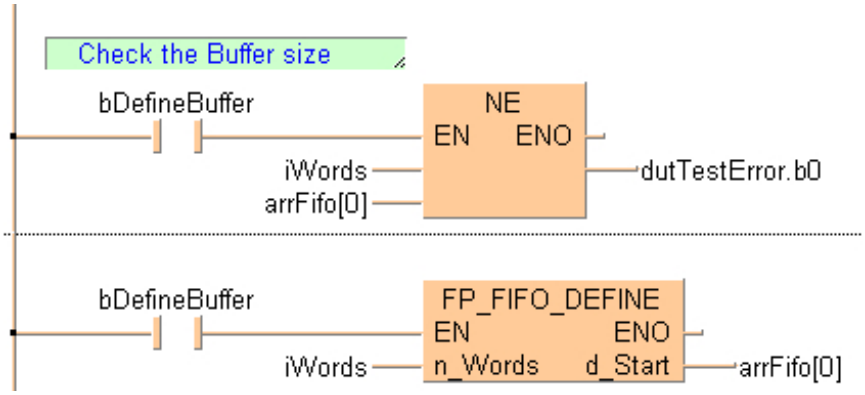
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bDefineBuffer | BOOL | FALSE |
| 1 | VAR | dutTestError | BOOL32_OVERLAPPING_DUT | |
| 2 | VAR | iWords | INT | 8 |
| 3 | VAR | arrFifo | ARRAY [0..11] OF INT | [12(0)] |

### POU body

When the variable **bDefineBuffer** is set to TRUE, the function is carried out.

### LD body



# FP_FIFO_READ

Read from FIFO buffer

This FP instruction reads the data from the starting data area of the FIFO (First-In-First-Out) buffer specified by **s_Start** and stores the data in the area specified by **d**.


## Parameters

Input

**s_Start** (WORD, INT, UINT)

Starting data area of the buffer

Output

**d** (WORD, INT, UINT)

Data area for storing the data read from FIFO buffer

## Remarks

- The variables **s_Start** and **d** have to be of the same data type.
- If this instruction is executed when the number of stored data is 0, an error occurs.
- If this instruction is executed when the reading pointer is equal to the writing pointer, an error occurs and reading is not carried out.
- If this instruction is executed when the reading pointer is indicating the final address in the FIFO buffer (**n_Words** defined by **FP_FIFO_DEFINE** minus 1), the reading pointer is set to 0.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the size (**n_Words**) of the buffer specified by **d_Start** is **n_Words** = 0, or when **n_Words** > 4096.
- if the number of stored data items of the buffer = 0.
- if the number of stored data items of the buffer > buffer size (**n_Words**).
- if the reading pointer > buffer size (**n_Words**).
- if the reading pointer is 4096 (16#1000) or higher after the data has been read.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the size (**n_Words**) of the buffer specified by **d_Start** is **n_Words** = 0, or when **n_Words** > 4096.
- if the number of stored data items of the buffer = 0.
- if the number of stored data items of the buffer > buffer size (**n_Words**).
- if the reading pointer > buffer size (**n_Words**).
- if the reading pointer is 4096 (16#1000) or higher after the data has been read.

## Example

### POU header

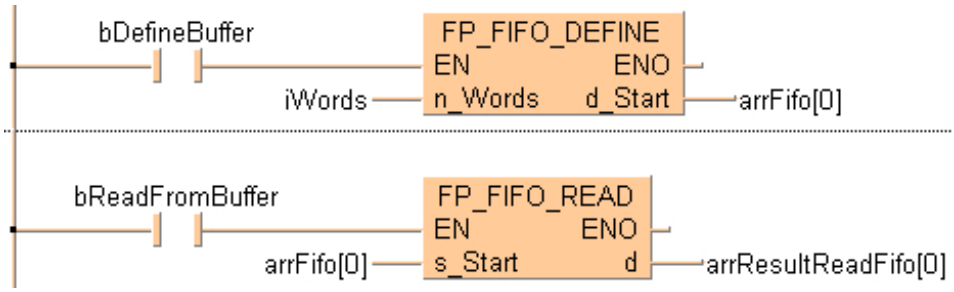
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bDefineBuffer | BOOL | FALSE |
| 1 | VAR | bWriteToBuffer | BOOL | FALSE |
| 2 | VAR | bReadFromBuffer | BOOL | FALSE |
| 3 | VAR | iWords | INT | 8 |
| 4 | VAR | arrFifo | ARRAY [0..11] OF INT | [12(0)] |
| 5 | VAR | arrResultReadFifo | ARRAY [0..7] OF WORD | [8(0)] |

### POU body

When the variable **bReadFromBuffer** is set to TRUE, the function is carried out.

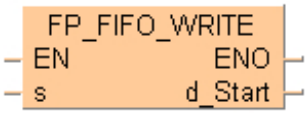
### LD body



# FP_FIFO_WRITE

Write to FIFO buffer

This FP instruction writes the data specified by **s** into the FIFO buffer specified by **d_Start**.




## Parameters

Input

**s** (WORD, INT, UINT)

Data area or equivalent constant for storing data to write in the FIFO buffer

Output

**d_Start** (WORD, INT, UINT)

Starting data area of the buffer

## Remarks

- The variables **s** and **d_Start** have to be of the same data type.
- If this instruction is executed when the FIFO buffer is full (number of stored data = **n_Words**, the size of the FIFO buffer defined by **FP_FIFO_DEFINE**), an error occurs and writing is not carried out.
- If this instruction is executed when the writing pointer is indicating the final address in the FIFO buffer (**n_Words** defined by **FP_FIFO_DEFINE**), the writing pointer will be set to 0.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the size (**n_Words**) of the buffer specified by **d_Start** is **n_Words** = 0, or when **n_Words** > 4096.
- if the number of stored data items of the buffer = 0.
- if the number of stored data items of the buffer > buffer size (**n_Words**).
- if the writing pointer > buffer size (**n_Words**).
- if the writing pointer is 4096 (16#1000) or higher after the data has been written.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the size (**n_Words**) of the buffer specified by **d_Start** is **n_Words** = 0, or when **n_Words** > 4096.
- if the number of stored data items of the buffer = 0.
- if the number of stored data items of the buffer > buffer size (**n_Words**).
- if the writing pointer > buffer size (**n_Words**).
- if the writing pointer is 4096 (16#1000) or higher after the data has been written.

## Example

### POU header

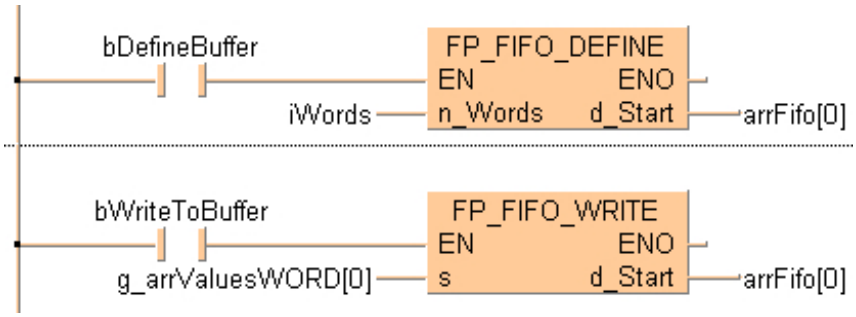
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bDefineBuffer | BOOL | FALSE |
| 1 | VAR | bWriteToBuffer | BOOL | FALSE |
| 2 | VAR | iWords | INT | 8 |
| 3 | VAR | arrFifo | ARRAY [0..11] OF INT | [12(0)] |
| 4 | VAR_EXTERNAL | g_arrValuesWORD | ARRAY [0..7] OF WORD | [16#0000,16#0001,... |

### POU body

When the variable **bWriteToBuffer** is set to TRUE, the function is carried out.

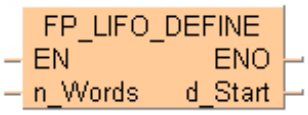
### LD body



# FP_LIFO_DEFINE

Define the LIFO buffer area

This FP instruction specifies the starting area **d1_Start** for the LIFO (Last-In-First-Out) buffer and the memory size **n_Words** of the LIFO buffer.




## Parameters

Input

**n_Words** (WORD, INT, UINT)

Specifies the memory size of the buffer
Values: 1–4096

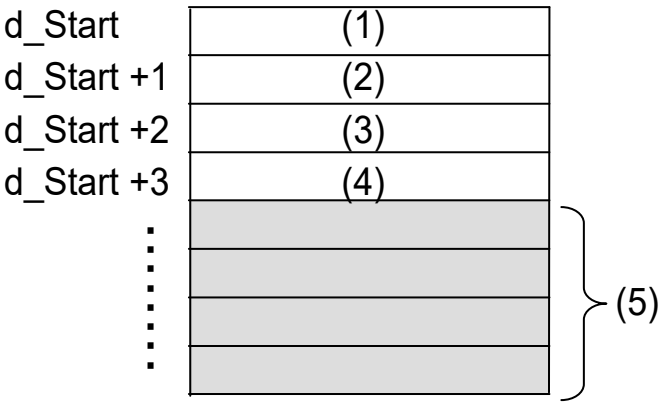
Output

**d_Start** (WORD, INT, UINT)

Starting data area of the buffer

## Remarks

Format of data buffer




(1) Buffer size
(2) Number of stored data
(3) Fixed to zero
(4) LIFO pointer
(5) Data buffer area (not cleared)

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a value specified for a parameter is outside the permissible range.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a value specified for a parameter is outside the permissible range.

## Example

## POU header

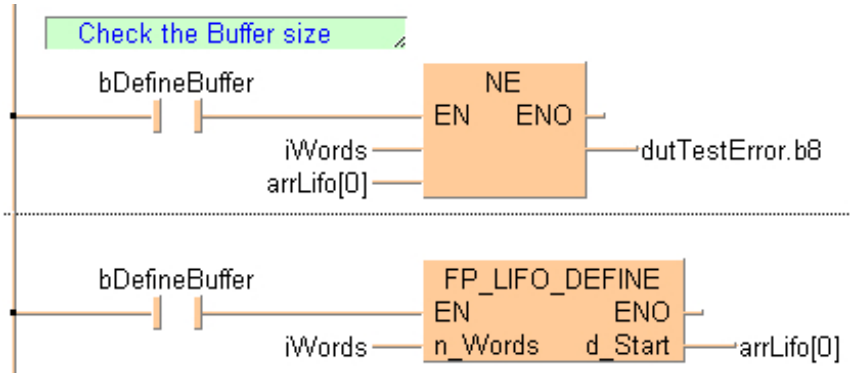
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bDefineBuffer | BOOL | FALSE |
| 1 | VAR | dutTestError | BOOL32_OVERLAPPING_DUT | |
| 2 | VAR | iWords | INT | 8 |
| 3 | VAR | arrLifo | ARRAY [0..11] OF INT | [12(0)] |

## POU body

When the variable **bDefineBuffer** is set to TRUE, the function is carried out.

## LD body



# FP_LIFO_READ

Read from LIFO buffer

This FP instruction reads the data from the starting data area of the LIFO (Last-In-First-Out) buffer specified by **s_Start** and stores the data in the area specified by **d**.


## Parameters

Input

**s_Start** (WORD, INT, UINT)

Starting data area of the buffer

Output

**d** (WORD, INT, UINT)

Data area for storing the data read from LIFO buffer

## Remarks

- The variables **s_Start** and **d** have to be of the same data type.
- If this instruction is executed when the number of stored data is 0, an error occurs.
- If this instruction is executed when the reading pointer is equal to the writing pointer, an error occurs and reading is not carried out.
- If this instruction is executed when the reading pointer is indicating the final address in the LIFO buffer (**n_Words** defined by **FP_LIFO_DEFINE** minus 1), the reading pointer is set to 0.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the size (**n_Words**) of the buffer specified by **d_Start** is **n_Words** = 0, or when **n_Words** > 4096.
- if the number of stored data items of the buffer = 0.
- if the number of stored data items of the buffer > buffer size (**n_Words**).
- if the reading pointer > buffer size (**n_Words**).
- if the reading pointer is 4096 (16#1000) or higher after the data has been read.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the size (**n_Words**) of the buffer specified by **d_Start** is **n_Words** = 0, or when **n_Words** > 4096.
- if the number of stored data items of the buffer = 0.
- if the number of stored data items of the buffer > buffer size (**n_Words**).
- if the reading pointer > buffer size (**n_Words**).
- if the reading pointer is 4096 (16#1000) or higher after the data has been read.

## Example

### POU header

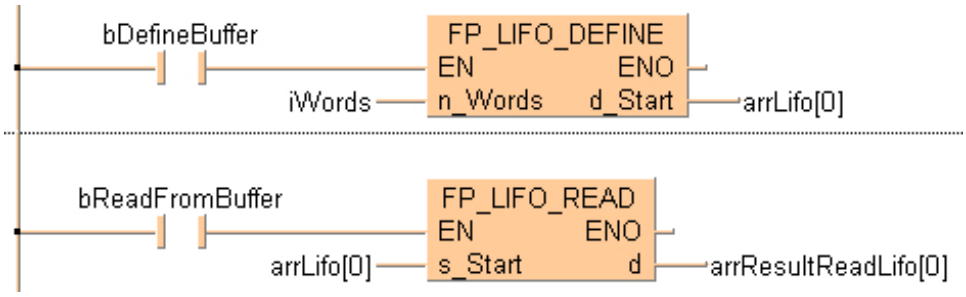
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bDefineBuffer | BOOL | FALSE |
| 1 | VAR | bReadFromBuffer | BOOL | FALSE |
| 2 | VAR | iWords | INT | 8 |
| 3 | VAR | arrLifo | ARRAY [0..11] OF INT | [12(0)] |
| 4 | VAR | arrResultReadLifo | ARRAY [0..7] OF WORD | [8(0)] |

### POU body

When the variable **bDefineBuffer** is set to TRUE, the function is carried out.

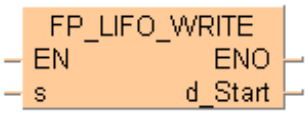
### LD body



# FP_LIFO_WRITE

Write to LIFO buffer

This FP instruction writes the data specified by **s** into the LIFO (Last-In-First-Out) buffer specified by **d_Start**.




## Remarks

- The variables **s** and **d_Start** have to be of the same data type.
- If this instruction is executed when the LIFO buffer is full (number of stored data = **n_Words**, the size of the LIFO buffer defined by **FP_LIFO_DEFINE**), an error occurs and writing is not carried out.
- If this instruction is executed when the writing pointer is indicating the final address in the LIFO buffer (**n_Words** defined by **FP_LIFO_DEFINE**), the writing pointer will be set to 0.

## Parameters

Input

**s** (WORD, INT, UINT)

Data area or equivalent constant for storing data to write in the LIFO buffer

Output

**d_Start** (WORD, INT, UINT)

Starting data area of the buffer

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the size (**n_Words**) of the buffer specified by **d_Start** is **n_Words** = 0, or when **n_Words** > 4096.
- if the number of stored data items of the buffer = 0.
- if the number of stored data items of the buffer > buffer size (**n_Words**).
- if the writing pointer > buffer size (**n_Words**).
- if the writing pointer is 4096 (16#1000) or higher after the data has been written.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the size (**n_Words**) of the buffer specified by **d_Start** is **n_Words** = 0, or when **n_Words** > 4096.
- if the number of stored data items of the buffer = 0.
- if the number of stored data items of the buffer > buffer size (**n_Words**).
- if the writing pointer > buffer size (**n_Words**).
- if the writing pointer is 4096 (16#1000) or higher after the data has been written.

## Example

### POU header

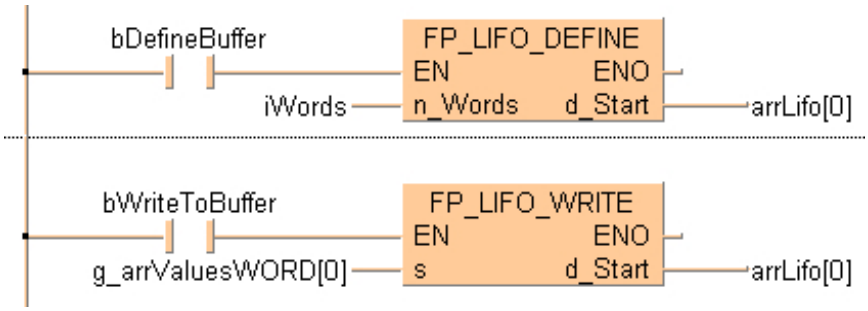
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bDefineBuffer | BOOL | FALSE |
| 1 | VAR | bWriteToBuffer | BOOL | FALSE |
| 2 | VAR | iWords | INT | 8 |
| 3 | VAR | arrLifo | ARRAY [0..11] OF INT | [12(0)] |
| 4 | VAR_EXTERNAL | g_arrValuesWORD | ARRAY [0..7] OF WORD | [16#0000... |

### POU body

When the variable **bWriteToBuffer** is set to TRUE, the function is carried out.

### LD body



## 15.2 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

Advantages of IEC instructions

Advantages of FP instructions

### 15.2.1 Data table analysis F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

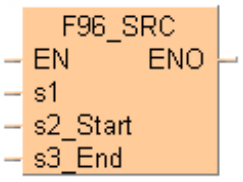
Advantages of IEC instructions

Advantages of FP instructions

# F96_SRC

Table data search (16-bit search)

Searches for the value that is the same as **s1** in the block of 16-bit areas specified by **s2_Start** (starting area) through **s3_End** (ending area) if the trigger **EN** is in the ON-state.




## Parameters

Input

**s1** (WORD, INT, UINT)

16-bit area or equivalent constant to store the value searched for

**s2_Start** (WORD, INT, UINT)

starting 16-bit area of the block

**s3_End** (WORD, INT, UINT)

ending 16-bit area of the block

## Remarks

- When the search operation is performed, the search results are stored as follows:
  - The number of data that is the same as **s1** is transferred to **sys_iNumberOfFoundMatches**.
  - The position the data is first found in, counting from the starting 16-bit area, is transferred to **sys_iPositionOfFirstMatch**.
- Be sure that **s2_Start** ≤ **s3_End**.
- The variables **s1**, **s2_Start** and **s3_End** have to be of the same data type.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s2_Start>s3_End**

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s2_Start>s3_End**

## Example

### POU header

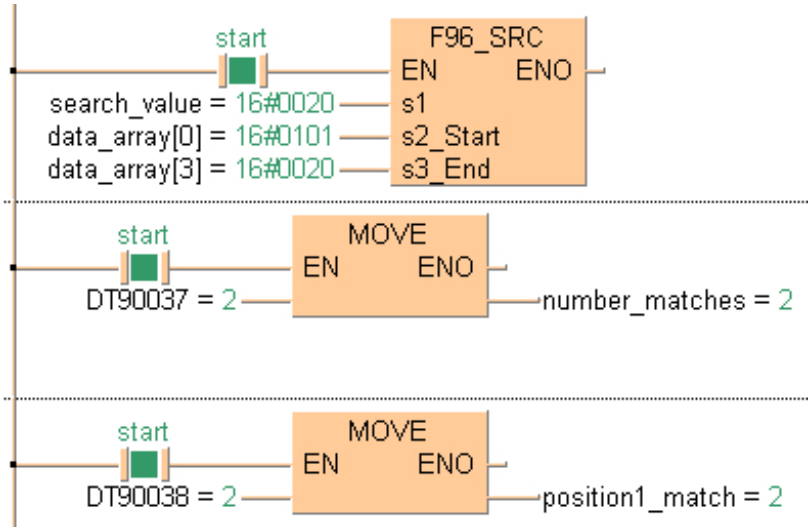
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the fuction |
| 1 | VAR | search_value | WORD | 16#20 | specifies the value to |
| 2 | VAR | data_array | ARRAY [0..3] OF WORD | [16#101,16#2A04,16#20,16#20] | 2 matches for 16#20<br>data_array[2] = 1st match |
| 3 | VAR | number_matches | INT | 0 | |
| 4 | VAR | position1_match | INT | 0 | |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

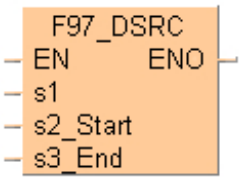
### LD body



# F97_DSRC

32-bit table data search

The function searches for the value specified at input **s1** in a block of 32-bit areas whose beginning is specified at input **s2_Start** and whose end is specified at input **s3_End**.




## Parameters

Input

**s1** (DWORD, DINT, UDINT, DATE, TOD, DT)

32-bit area or equivalent constant to store the value searched for

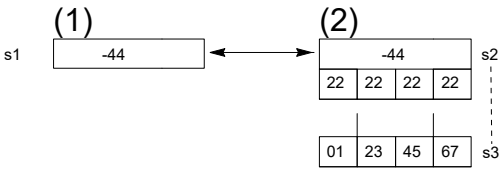
**s2_Start** (DWORD, DINT, UDINT, DATE, TOD, DT)

starting 32-bit area of the block

**s3_End** (DWORD, DINT, UDINT, DATE, TOD, DT)

ending 32-bit area of the block

## Remarks




(1) Value searched for
(2) 32-bit table data

- The number of data items that match **s1** is stored in special data register DT90037 (**sys_iNumberOfFoundMatches**).
- The relative position of the first matching data item, counting from the starting 32-bit area **s2_Start**, is stored in special data register DT90038 (**sys_iPositionOfFirstMatch**).
- The adresses of the variables at inputs **s2_Start** and **s3_End** must be of the same adress type.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s2_Start>s3_End**

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s2_Start>s3_End**

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

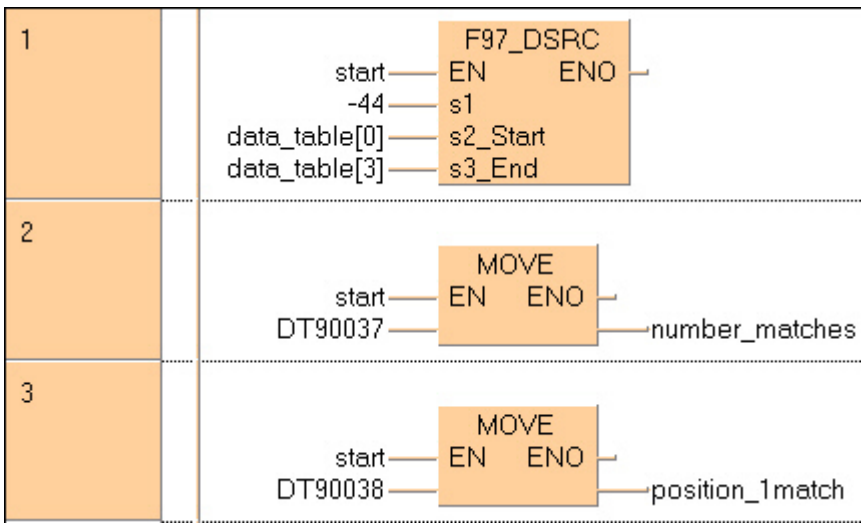
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the funktion |
| 1 | VAR | data_table | ARRAY [0..3] OF DINT | [-44,222222,-44,12345] | Arbitrarily large data field |
| 2 | VAR | number_matches | INT | 0 | result: here 2 |
| 3 | VAR | position_1match | INT | 0 | result: here 0 |

## POU body

When the variable **start** is set to TRUE, the function is carried out.

Instead of using an input variable in this example, a constant (-44) is assigned to input s1. The **result** is stored in special data registers DT90037 and DT90038. The two E_MOVE functions copy the results to the two variables **number_matches** and **position_1match**.

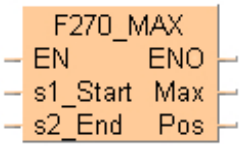
## LD body



# F270_MAX

Maximum value search in 16-bit data table

The function searches for the maximum value and its position in a 16-bit data table.




## Parameters

Input

**s1_Start** (WORD, INT, UINT)

Beginning of data table

**s2_End** (WORD, INT, UINT)

End of data table

Output

**Max** (INT)

maximum value

**Pos** (INT)

position of maximum value

## Remarks

- Input **s1_Start** specifies the starting area of the data table, and **s2_End** specifies the end. The maximum value is returned at output **Max** and its position at output **Pos**.
- The position **Pos** is relative to the position at the beginning of the data table to the first occurrence of the maximum value.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

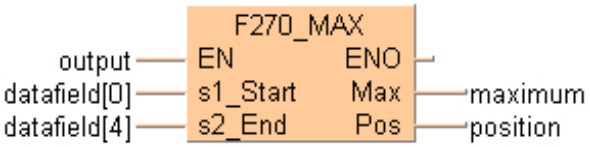
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | data_field | ARRAY [0..4] OF INT | [2,3,6,-3,1] | Arbitrarily large data field |
| 2 | VAR | maximum_value | INT | 0 | result: here 6 |
| 3 | VAR | position | INT | 0 | result: here 2 |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

It searches for the maximum value and its position in the **data_field**. The **result** is here: **maximum_value** = 6 and **position** = 2.

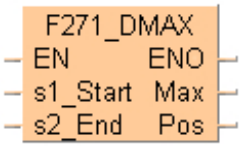
### LD body



# F271_DMAX

Maximum value search in 32-bit data table

The function searches for the maximum value and its position in a 32-bit data table.




## Parameters

Input

**s1_Start** (DWORD, DINT, UDINT, DATE, TOD, DT)

Beginning of data table

**s2_End** (DWORD, DINT, UDINT, DATE, TOD, DT)

End of data table

Output

**Max** (DINT)

maximum value

**Pos** (INT)

position of maximum value

## Remarks

- Input **s1_Start** specifies the starting area of the data table, and **s2_End** specifies the end. The maximum value is returned at output **Max** and its position at output **Pos**.
- The position **Pos** is relative to the position at the beginning of the data table to the first occurrence of the maximum value.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

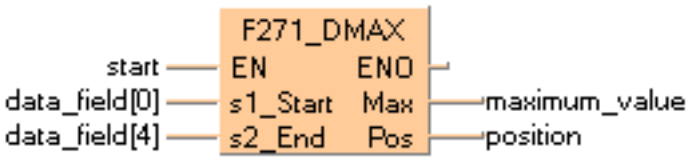
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | data_field | ARRAY [0..4] OF DINT | [2,3,222222,-333333,1] | Arbitrarily large data field |
| 2 | VAR | maximum_value | DINT | 0 | result: here 222222 |
| 3 | VAR | position | INT | 0 | result: here 2 |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

It searches for the maximum value and its position in the **data_field**. The **result** is here: **maximum_value** = 222222 and **position** = 2.

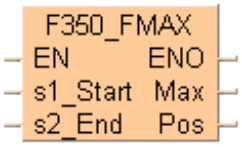
### LD body



# F350_FMAX

Maximum value search in real number data table (floating point data)

The function searches for the maximum value and its position in a floating point data table.




## Parameters

Input

**s1_Start** (REAL)

Beginning of data table

**s2_End** (REAL)

End of data table

Output

**Max** (REAL)

maximum value

**Pos** (INT)

position of maximum value

## Remarks

- Input **s1_Start** specifies the starting area of the data table, and **s2_End** specifies the end. The maximum value is returned at output **Max** and its position at output **Pos**.
- The position **Pos** is relative to the position at the beginning of the data table to the first occurrence of the maximum value.
- If more than one maximum value is found, the first one found beginning from the starting address specified at **s1_Start** is stored in **Max**.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the address of the variable at input **s1_Start** > **s2_End**.
- if the address areas of **s1_Start** and **s2_End** are different.
- if the floating point values exceed the processing range.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the address of the variable at input **s1_Start** > **s2_End**.
- if the address areas of **s1_Start** and **s2_End** are different.
- if the floating point values exceed the processing range.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

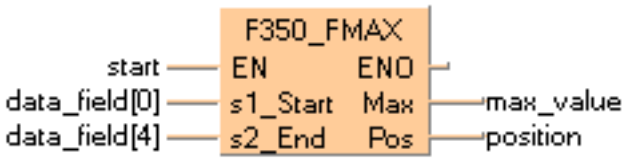
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | data_field | ARRAY [0...4] OF REAL | [2.0,3.45,-6.91,5.44,1.3] | Arbitrarily large data field |
| 2 | VAR | max_value | REAL | 0.0 | result: here 5.44 |
| 3 | VAR | position | INT | 0 | result: here 3 |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

It then searches the **data_field** for a maximum value and its position. The **result** here is: **max_value** = 5.44 and **position** = 3.

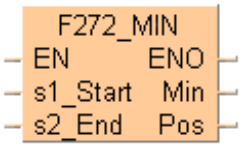
### LD body



# F272_MIN

Minimum value search in 16-bit data table

The function searches for the minimum value and its position in a 16-bit data table.




## Parameters

Input

**s1_Start** (WORD, INT, UINT)

Beginning of data table

**s2_End** (WORD, INT, UINT)

End of data table

Output

**Min** (INT)

Minimum value

**Pos** (INT)

Position of minimum value

## Remarks

- Input **s1_Start** specifies the starting area of the data table, and **s2_End** specifies the end. The minimum value is returned at output **Min** and its position at output **Pos**.
- The position **Pos** is relative to the position at the beginning of the data table to the first occurrence of the minimum value.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

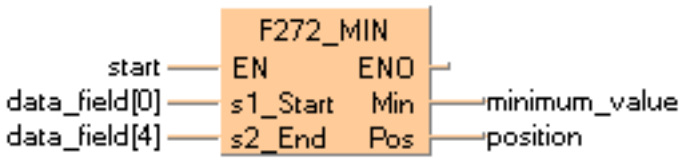
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | data_field | ARRAY [0..4] OF INT | [2,3,6,-3,1] | Arbitrarily large data field |
| 2 | VAR | minimum_value | INT | 0 | result: here -3 |
| 3 | VAR | position | INT | 0 | result: here 3 |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

It searches for the **minimum** value and its position in the **data_field**. The **result** is here: **minimum_value** = -3 and **position** = 3.

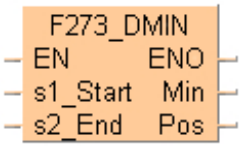
### LD body



# F273_DMIN

Minimum value search in 32-bit data table

The function searches for the minimum value and its position in a 32-bit data table.




## Parameters

Input

**s1_Start** (DWORD, DINT, UDINT, DATE, TOD, DT)

Beginning of data table

**s2_End** (DWORD, DINT, UDINT, DATE, TOD, DT)

End of data table

Output

**Min** (DINT)

minimum value

**Pos** (INT)

position of minimum value

## Remarks

- Input **s1_Start** specifies the starting area of the data table, and **s2_End** specifies the end. The minimum value is returned at output **Min** and its position at output **Pos**.
- The position **Pos** is relative to the position at the beginning of the data table to the first occurrence of the minimum value.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

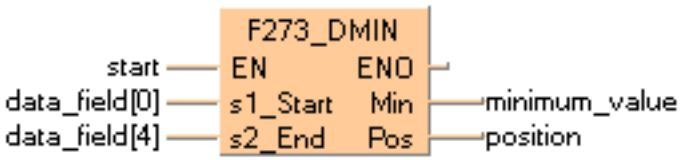
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | data_field | ARRAY [0..4] OF DINT | [2,3,222222,-333333,1] | Arbitrarily large data field |
| 2 | VAR | minimum_value | DINT | 0 | result: here -333333 |
| 3 | VAR | position | INT | 0 | result: here 3 |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

It searches for the **minimum** value and its position in the **data_field**. The **result** is here: **minimum_value** = -333333 and **position** = 3.

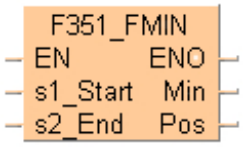
### LD body



# F351_FMIN

Minimum value search in real number data table (floating point data)

The function searches for the minimum value and its position in a floating point data table.




## Parameters

Input

**s1_Start** (REAL)

Beginning of data table

**s2_End** (REAL)

End of data table

Output

**Min** (REAL)

minimum value

**Pos** (INT)

position of minimum value

## Remarks

- Input **s1_Start** specifies the starting area of the data table, and **s2_:End** specifies the end. The minimum value is returned at output **Min** and its position at output **Pos**.
- If more than one minimum value is found, the first one found beginning from the starting address specified at **s1_Start** is stored in **Min**.
- The address of the minimum value at output **Pos** is relative to the beginning address in the data table as specified at input **s1_Start**.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the address of the variable at input **s1_Start** > **s2_End**.
- if the address areas of **s1_Start** and **s2_End** are different.
- if the floating point values exceed the processing range.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the address of the variable at input **s1_Start** > **s2_End**.
- if the address areas of **s1_Start** and **s2_End** are different.
- if the floating point values exceed the processing range.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

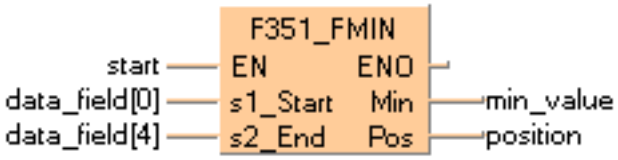
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | data_field | ARRAY [0..4] OF REAL | [2.0,3.45,-6.91,5.44,1.3] | Arbitrarily large data field |
| 2 | VAR | min_value | REAL | 0.0 | result: here -6.91 |
| 3 | VAR | position | INT | 0 | result: here 2 |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

It then searches the **data_field** for a **minimum** value and its position. The **result** here is: **min_value** = 6.91 and **position** = 2.

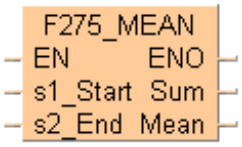
### LD body



# F275_MEAN

Total and mean numbers calculation in 16-bit data table

This function calculates the sum and the arithmetic mean of numbers (both with +/- signs) in the specified 16-bit data table.




## Parameters

Input

**s1_Start** (WORD, INT, UINT)

starting area of data table

**s2_End** (WORD, INT, UINT)

ending area of data table

Output

**Sum** (DINT)

sum of all elements in data table area specified

**Mean** (INT)

mean of all elements in data table area specified

## Remarks

Input **s1_Start** specifies the starting area of the data table, and **s2_End** specifies the end. The sum of all elements in the data table is returned at output **Sum** and the arithmetic mean of all elements in the data table is returned at output **Mean**. The arithmetic mean is rounded off if it is not a whole number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the total value range overflows or underflows the 16-bit range.

## Example

### POU header

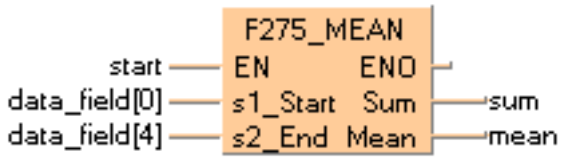
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | data_field | ARRAY [0..4] OF INT | [2,3,6,-3,1] | Arbitrarily large data field |
| 2 | VAR | sum | DINT | 0 | result: here 9 |
| 3 | VAR | mean | INT | 0 | result: here 1 |

### POU body

When the variable **start** is set to TRUE, the function is carried out. The function calculates the sum of all elements of the data table (`sum = 4 + 3 + 8 + (-2) + 1 + (-6) = 8`) and writes the result (in this case 8) to the variable **sum**. Additionally, the function calculates the arithmetic mean of all elements of the data table `(mean = sum/6 = (4 + 3 + 8 + (-2) + 1 + (-6)) / 6 = 1.333`) and writes the rounded-off number (in this case 1) to the variable **mean**.

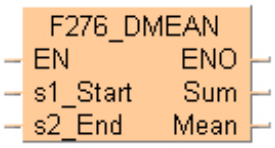
### LD body



# F276_DMEAN

Total and mean numbers calculation in 32-bit data table

This function calculates the sum and the arithmetic mean of numbers (both with +/- signs) in the specified 32-bit data table.




## Parameters

Input

**s1_Start** (DWORD, DINT, UDINT, DATE, TOD, DT)

starting area of data table

**s2_End** (DWORD, DINT, UDINT, DATE, TOD, DT)

ending area of data table

Output

**Sum** (ARRAY [0..1] OF DINT)

sum of all elements in data table area specified

**Mean** (DINT)

mean of all elements in data table area specified

## Remarks

Input **s1_Start** specifies the starting area of the data table, and **s2_End** specifies the end. The sum of all elements in the data table is returned at output **Sum** and the arithmetic mean of all elements in the data table is returned at output **Mean**. The arithmetic mean is rounded off if it is not a whole number.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_blsCarry** (turns to TRUE for one scan)

- if the total value range overflows or underflows the 32-bit range.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

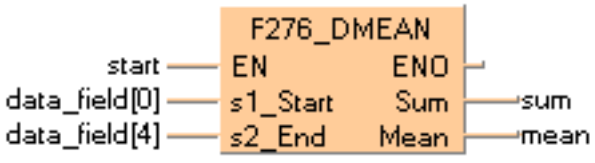
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | output | BOOL | FALSE | activates the function |
| 1 | VAR | data_field | ARRAY [0..4] OF DINT | [2,3,222222,-333333,1] | Arbitrarily large data field |
| 2 | VAR | sum | ARRAY [0..1] OF DINT | [2(0)] | result: here |
| 3 | VAR | mean | DINT | 0 | result: here -22221 |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

The function calculates the sum of all elements of ARRAY **data_field** (sum = 2 + 3 + 222222 + (-333333) + 1 = -111105) and transfers the **result** to the variable **sum**. In addition, the function calculates the mean (mean = sum/5 = -111105/5 = -22221) and transfers the **result** to the variable **mean**.

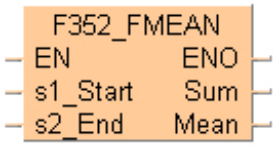
### LD body



# F352_FMEAN

Total and mean numbers calculation in floating point data table

This function calculates the sum and the arithmetic mean (both with +/- signs) of floating point values in the specified 32-bit data table.




## Parameters

Input

**s1_Start** (REAL)

starting area of data table

**s2_End** (REAL)

ending area of data table

Output

**Sum** (REAL)

sum of all elements in data table area specified

**Mean** (REAL)

mean of all elements in data table area specified

## Remarks

Input **s1_Start** specifies the starting area of the data table, and **s2_End** specifies the end. The sum of all elements in the data table is returned at output **Sum** and the arithmetic mean of all elements in the data table is returned at output **Mean**.

Instead of using this F instruction, we recommend using the corresponding FP7 instruction:

- **FP_DATA_MEAN_SUM_UINT** (page 998)
- **FP_DATA_MEAN_SUM_UDINT0** (page 996)
- **FP_DATA_MEAN_SUM_REAL** (page 994)
- **FP_DATA_MEAN_SUM_INT** (page 992)
- **FP_DATA_MEAN_SUM_DINT0** (page 990)

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the address of the variable at input **s1_Start** > **s2_End**.
- if the address areas are different.
- if the floating point values exceed the processing range.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the address of the variable at input **s1_Start** > **s2_End**.
- if the address areas are different.
- if the floating point values exceed the processing range.

**sys_bIsCarry** (turns to TRUE for one scan)

- if the total value range overflows or an underflow.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

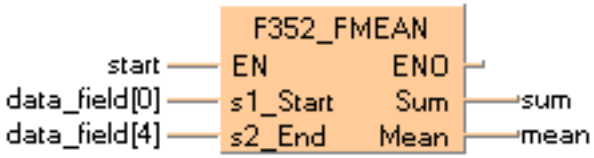
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | data_field | ARRAY [0..4] OF REAL | [2.0,3.45,-6.91,5.44,1.3] | Arbitrarily large data field |
| 2 | VAR | sum | REAL | 0.0 | result: here 5.28 |
| 3 | VAR | mean | REAL | 0.0 | result: here 1.056 |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

It calculates the **sum** `= 2.0 + 3.45 + (-6.91) + 5.44 + 1.3 = 5.28` and the **mean** `= Sum/5 = 5.28/5 = 1.056` of the elements of the **data_field**.

### LD body



### 15.2.2 Data table manipulation F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

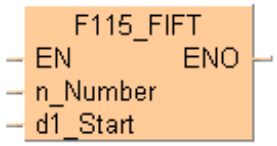
Advantages of IEC instructions

Advantages of FP instructions

# F115_FIFT

FIFO buffer area definition

F115 specifies the starting area **d1_Start** for the FIFO (First-In-First-Out) buffer and the memory size **n_Number** of the FIFO buffer.




## Parameters

Input

**n_Number** (INT)

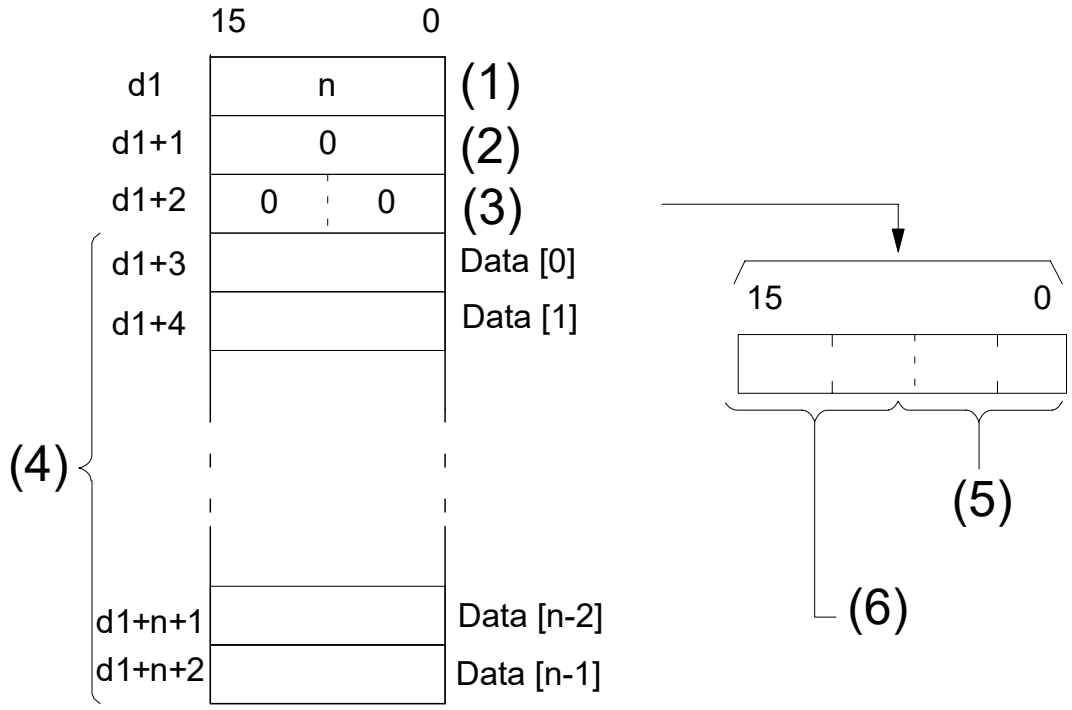
specifies the memory size of FIFO buffer
range: 1–256

**d1_Start** (WORD, INT, UINT)

starting 16-bit area of FIFO buffer

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_FIFO_DEFINE**, **FP_LIFO_DEFINE**
- Definition of the area using the FIFT instruction should be carried out only once, before writing to or reading from the FIFO buffer. When the FIFT instruction is executed, the FIFO buffer area is defined as follows:




(1) Memory size of FIFO buffer (n)
(2) Number of stored data items (words), written and not read
(3) FIFO pointer
(4) Data storage area (n words)
(5) Writing pointer (0–255/16#00–16#FF)
(6) Reading pointer (0–255/16#00–16#FF)

- When the FIFT instruction is executed, the following are stored as default values: **d1_Start** = **n_Number** (the value specified by the FIFT instruction), **d1_Start** + 1 = 0, and **d1_Start** + 2 = 16#0000.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if **n_Number** = 0
- if **n_Number** > 256
- if the area specified by **n_Number** exceeds the limit

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if **n_Number** = 0
- if **n_Number** > 256
- if the area specified by **n_Number** exceeds the limit

## Example

This example illustrates the FIFO buffer by incorporating the functions **F115_FIFT**, **F116_FIFR** and **F117_FIFW**.

### DUT

FIFO_n_WORD

| | Identifier | Type | Initial |
|---|---|---|---|
| 1 | iSize | INT | 0 |
| 2 | iNumber | INT | 0 |
| 3 | wPositions | WORD | 0 |
| 4 | awData | ARRAY [0..12] OF WORD | [13(0)] |

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

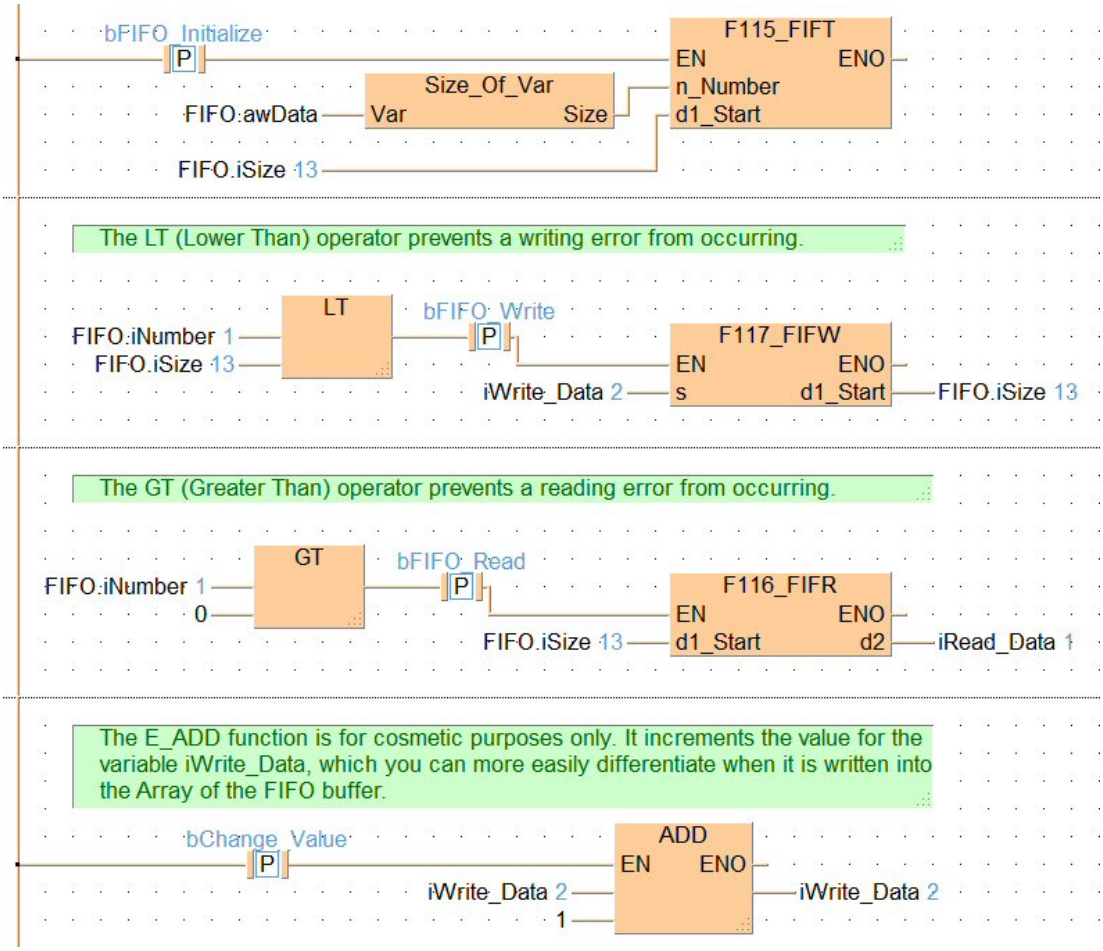
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | FIFO | FIFO_n_WORD | |
| 2 | VAR | iRead_Data | INT | 0 |
| 3 | VAR | iWrite_Data | INT | 1 |
| 4 | VAR | bFIFO_Initialize | BOOL | FALSE |
| 5 | VAR | bFIFO_Write | BOOL | FALSE |
| 6 | VAR | bFIFO_Read | BOOL | FALSE |
| 7 | VAR | bChange_Value | BOOL | FALSE |

### POU body

The example below illustrates the status of the buffer after **bFIFO_Write** has been enabled twice and **bFIFO_Read** once. When **bFIFO_Write** was activated the first time, the value 1 was written into **FIFO.awData[0]**. When **bFIFO_Read** was enabled, **iRead_Data** then read this value. When **bFIFO_Write** was enabled the second time, the writing pointer was incremented by one and the value 2 written into **FIFO.awData[1]**. see Entry Data Monitor 1

| | Identifier | Value | FP address |
|---|---|---|---|
| 1 | F115_FIFT_F116_F117 | | |
| 2 | FIFO | | |
| 3 | iSize | 13 | DT3293 |
| 4 | iNumber | 1 | DT3294 |
| 5 | wPositions | 16#0102 | DT3295 |
| 6 | awData | | |
| 7 | [0] | 16#0001 | DT3296 |
| 8 | [1] | 16#0002 | DT3297 |
| 9 | [2] | 16#0000 | DT3298 |
| 10 | [3] | 16#0000 | DT3299 |
| 11 | [4] | 16#0000 | DT3300 |
| 12 | [5] | 16#0000 | DT3301 |
| 13 | [6] | 16#0000 | DT3302 |
| 14 | [7] | 16#0000 | DT3303 |
| 15 | [8] | 16#0000 | DT3304 |
| 16 | [9] | 16#0000 | DT3305 |
| 17 | [10] | 16#0000 | DT3306 |
| 18 | [11] | 16#0000 | DT3307 |
| 19 | [12] | 16#0000 | DT3308 |
| 20 | iRead_Data | 1 | DT3309 |
| 21 | iWrite_Data | 2 | DT3310 |
| 22 | bFIFO_Initialize | TRUE | R264 |
| 23 | bFIFO_Write | TRUE | R265 |
| 24 | bFIFO_Read | TRUE | R266 |
| 25 | bChange_Value | TRUE | R267 |

## LD body


bFIFO_Initialize
P
F115_FIFT
EN
ENO
Size_Of_Var
FIFO.awData
Var
Size
n_Number
d1_Start
FIFO.iSize 13
The LT (Lower Than) operator prevents a writing error from occurring.
LT
bFIFO_Write
FIFO.iNumber 1
FIFO.iSize 13
F117_FIFW
iWrite_Data 2
s
d1_Start
FIFO.iSize 13
The GT (Greater Than) operator prevents a reading error from occurring.
GT
bFIFO_Read
FIFO.iNumber 1
0
F116_FIFR
FIFO.iSize 13
d1_Start
d2
iRead_Data 1
The E_ADD function is for cosmetic purposes only. It increments the value for the variable iWrite_Data, which you can more easily differentiate when it is written into the Array of the FIFO buffer.
bChange_Value
ADD
iWrite_Data 2
1
iWrite_Data 2

# F116_FIFR

Read from FIFO buffer

**F116_FIFR** reads the data **d1** from the FIFO (First-In-First-Out) buffer and stores the data in area specified by **d2**.


## Parameters

Input

**d1_Start** (WORD, INT, UINT)

starting 16-bit area of FIFO buffer

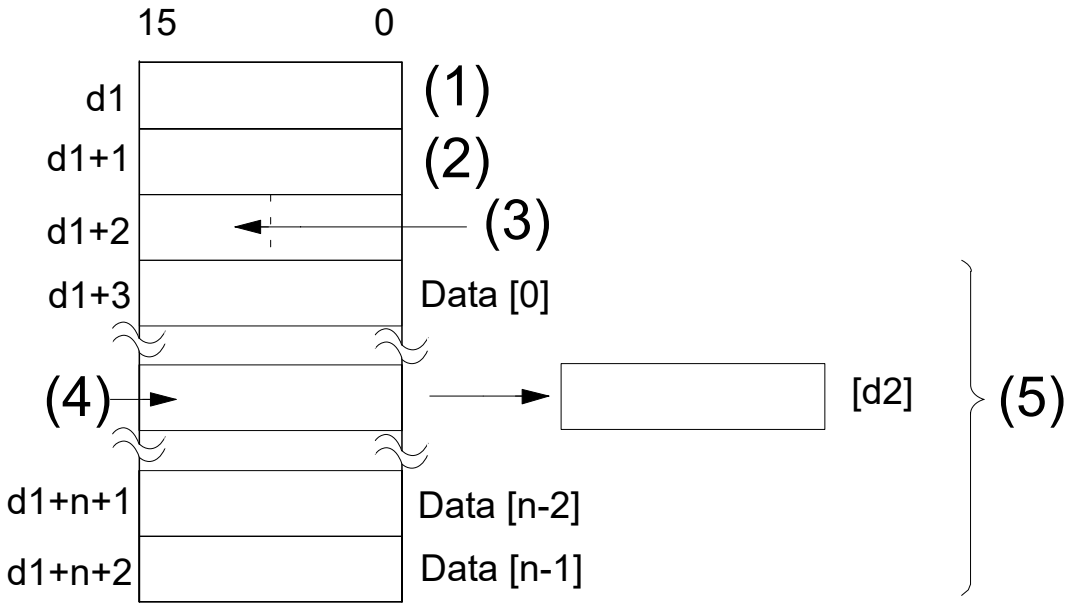
Output

**d2** (WORD, INT, UINT)

16-bit area for storing data read from FIFO buffer

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_FIFO_READ**, **FP_LIFO_READ**
- Reading of data is done starting from the address specified by the reading pointer when the instruction is executed.




(1) Memory size of FIFO buffer (n)
(2) Number of stored data items (words), written and not read
(3) Reading pointer in upper byte
(4) Reading pointer
(5) Data storage area (n words)

- (0), (n–2) and (n–1) are addresses assigned to the data storage area.
- n is the value specified by the **F115_FIFT** instruction.
- The reading pointer is stored in the upper eight bits of the third word of the FIFO buffer area. The actual address is the value of the leading address in the FIFO buffer area specified by **d1** plus 3, plus the value of reading pointer (the value of which only the first byte is a decimal value).
- When the reading is executed, 1 is subtracted from the number of stored data items, and the reading pointer is incremented by 1, or reset to zero if the reading pointer pointed to the final element.

Note

- An error occurs if this is executed when the number of stored data items is 0 or when the reading pointer is equal to the writing pointer.
- Reading is only carried out when the reading pointer is not equal to the writing pointer.
- If this is executed when the reading pointer is indicating the final address in the FIFO buffer (the **n** defined by the FIFO instruction minus 1), the reading pointer is set to 0.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the size (n) of the FIFO specified by **d1** is **n** = 0, or when **n** > 256.
- if the number of stored data items of the FIFO = 0.
- if the number of stored data items of the FIFO > FIFO size (n).

- if the final address of the FIFO based on the FIFO size (n) exceeds the area.
- if the FIFO reading pointer > FIFO size (n).
- if the FIFO reading pointer is 256 (16#100) or higher after the data has been read.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the size (n) of the FIFO specified by **d1** is **n** = 0, or when **n** > 256.
- if the number of stored data items of the FIFO = 0.
- if the number of stored data items of the FIFO > FIFO size (n).
- if the final address of the FIFO based on the FIFO size (n) exceeds the area.
- if the FIFO reading pointer > FIFO size (n).
- if the FIFO reading pointer is 256 (16#100) or higher after the data has been read.

## Example

This example illustrates the FIFO buffer by incorporating the functions **F115_FIFT**, **F116_FIFR** and **F117_FIFW**.

### DUT

FIFO_n_WORD

| | Identifier | Type | Initial |
|---|---|---|---|
| 1 | iSize | INT | 0 |
| 2 | iNumber | INT | 0 |
| 3 | wPositions | WORD | 0 |
| 4 | awData | ARRAY [0..12] OF WORD | [13(0)] |

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

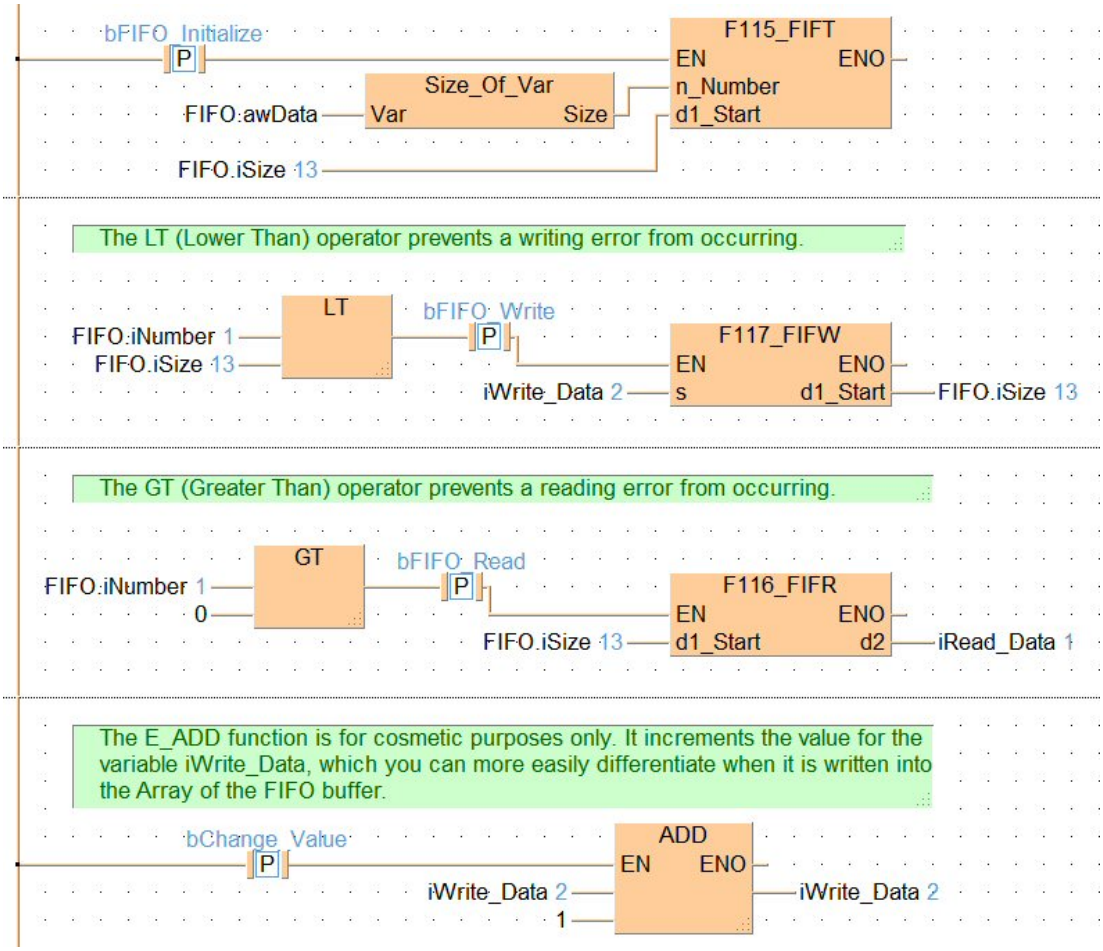
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | FIFO | FIFO_n_WORD | |
| 2 | VAR | iRead_Data | INT | 0 |
| 3 | VAR | iWrite_Data | INT | 1 |
| 4 | VAR | bFIFO_Initialize | BOOL | FALSE |
| 5 | VAR | bFIFO_Write | BOOL | FALSE |
| 6 | VAR | bFIFO_Read | BOOL | FALSE |
| 7 | VAR | bChange_Value | BOOL | FALSE |

## POU body

The example below illustrates the status of the buffer after **bFIFO_Write** has been enabled twice and **bFIFO_Read** once. When **bFIFO_Write** was activated the first time, the value 1 was written into **FIFO.awData[0]**. When **bFIFO_Read** was enabled, **iRead_Data** then read this value. When **bFIFO_Write** was enabled the second time, the writing pointer was incremented by one and the value 2 written into **FIFO.awData[1]**. see Entry Data Monitor 1

| | Identifier | Value | FP address |
|---|---|---|---|
| 1 | F115_FIFT_F116_F117 | | |
| 2 | FIFO | | |
| 3 | iSize | 13 | DT3293 |
| 4 | iNumber | 1 | DT3294 |
| 5 | wPositions | 16#0102 | DT3295 |
| 6 | awData | | |
| 7 | [0] | 16#0001 | DT3296 |
| 8 | [1] | 16#0002 | DT3297 |
| 9 | [2] | 16#0000 | DT3298 |
| 10 | [3] | 16#0000 | DT3299 |
| 11 | [4] | 16#0000 | DT3300 |
| 12 | [5] | 16#0000 | DT3301 |
| 13 | [6] | 16#0000 | DT3302 |
| 14 | [7] | 16#0000 | DT3303 |
| 15 | [8] | 16#0000 | DT3304 |
| 16 | [9] | 16#0000 | DT3305 |
| 17 | [10] | 16#0000 | DT3306 |
| 18 | [11] | 16#0000 | DT3307 |
| 19 | [12] | 16#0000 | DT3308 |
| 20 | iRead_Data | 1 | DT3309 |
| 21 | iWrite_Data | 2 | DT3310 |
| 22 | bFIFO_Initialize | TRUE | R264 |
| 23 | bFIFO_Write | TRUE | R265 |
| 24 | bFIFO_Read | TRUE | R266 |
| 25 | bChange_Value | TRUE | R267 |

## LD body


bFIFO_Initialize
P
F115_FIFT
EN
ENO
Size_Of_Var
FIFO.awData
Var
Size
n_Number
d1_Start
FIFO.iSize 13
The LT (Lower Than) operator prevents a writing error from occurring.
LT
FIFO.iNumber 1
FIFO.iSize 13
bFIFO_Write
F117_FIFW
iWrite_Data 2
s
d1_Start
The GT (Greater Than) operator prevents a reading error from occurring.
GT
FIFO.iNumber 1
0
bFIFO_Read
F116_FIFR
d2
iRead_Data 1
The E_ADD function is for cosmetic purposes only. It increments the value for the variable iWrite_Data, which you can more easily differentiate when it is written into the Array of the FIFO buffer.
bChange_Value
ADD
1

# F117_FIFW

Write to FIFO buffer

F/P117 writes the data specified by **s** into the FIFO buffer specified by **d1**.


## Parameters

Input

**s** (WORD, INT, UINT)

16-bit area or equivalent constant for storing data to write in the FIFO buffer

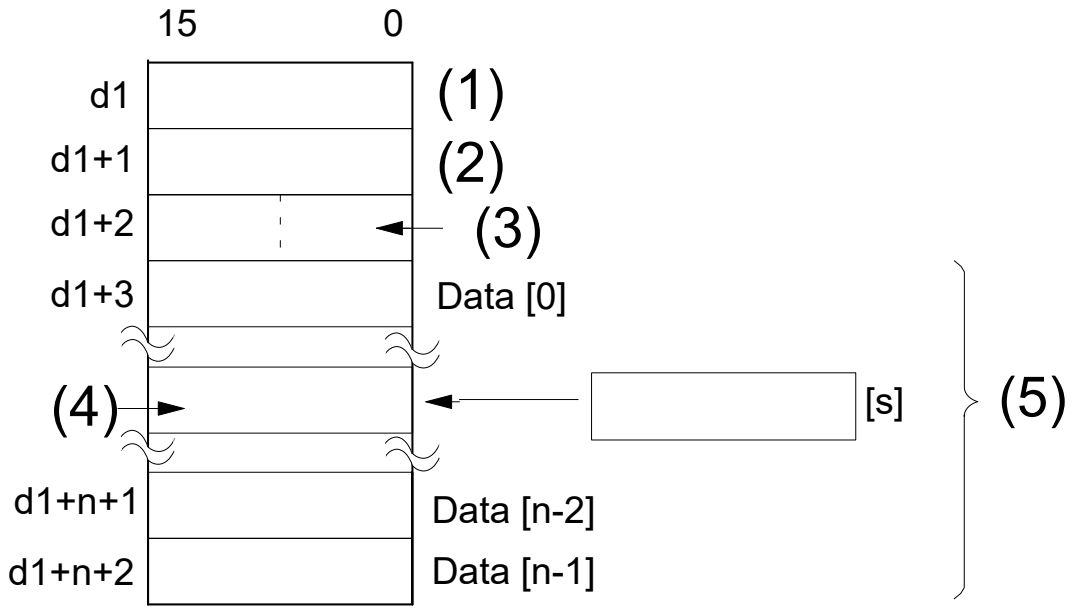
Output

**d1_Start** (WORD, INT, UINT)

starting 16-bit area of FIFO buffer

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_FIFO_WRITE** (page 1018), **FP_LIFO_WRITE** (page 1024)
- The specified data is written to the address indicated by the writing pointer when the instruction is executed.




(1) Memory size of FIFO buffer (n)
(2) Number of stored data items (words), written and not read
(3) Reading pointer in upper byte
(4) Reading pointer
(5) Data storage area (n words)

- (0), (n-2) and (n-1) are addresses assigned to the data storage area.
- **n** is the value specified by the **F115_FIFT** (page 1051) instruction.
- The writing pointer is stored in the lower eight bits of the third word of the FIFO buffer area, and is indicated by a relative position in the data storage area. The actual address to which data is being written is specified by **d1** plus the offset 3 plus the value of the writing pointer (the value of which only the lower byte is a decimal value).
- When the writing is executed, 1 is added to the number of stored data items, and the writing pointer is incremented by 1, or reset to zero if the writing pointer pointed to the final element.
- The variables **s** and **d1_Start** have to be of the same data type.

Note

- An error occurs if this is executed when the FIFO buffer is full (the number of stored data items = the size **n** of the FIFO defined by the FIFT instruction). Writing is inhibited.
- If this is executed when the writing pointer is indicating the final address in the FIFO buffer (the "n" value defined by the FIFT instruction), the writing pointer will be set to 0.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the size (n) of the FIFO specified by **d1_Start** is **n** = 0, or when **n** > 256.
- if the number of stored data items of the FIFO = 0.
- if the number of stored data items of the FIFO > FIFO size (n).

- if the final address of the FIFO based on the FIFO size (n) exceeds the area.
- if the FIFO writing pointer > FIFO size (n).
- if the FIFO writing pointer is 256 (16#100) or higher after the data has been written.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the size (n) of the FIFO specified by **d1_Start** is **n** = 0, or when **n** > 256.
- if the number of stored data items of the FIFO = 0.
- if the number of stored data items of the FIFO > FIFO size (n).
- if the final address of the FIFO based on the FIFO size (n) exceeds the area.
- if the FIFO writing pointer > FIFO size (n).
- if the FIFO writing pointer is 256 (16#100) or higher after the data has been written.

## Example

This example illustrates the FIFO buffer by incorporating the functions **F115_FIFT**, **F116_FIFR** and **F117_FIFW**.

### DUT

FIFO_n_WORD

| | Identifier | Type | Initial |
|---|---|---|---|
| 1 | iSize | INT | 0 |
| 2 | iNumber | INT | 0 |
| 3 | wPositions | WORD | 0 |
| 4 | awData | ARRAY [0..12] OF WORD | [13(0)] |

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

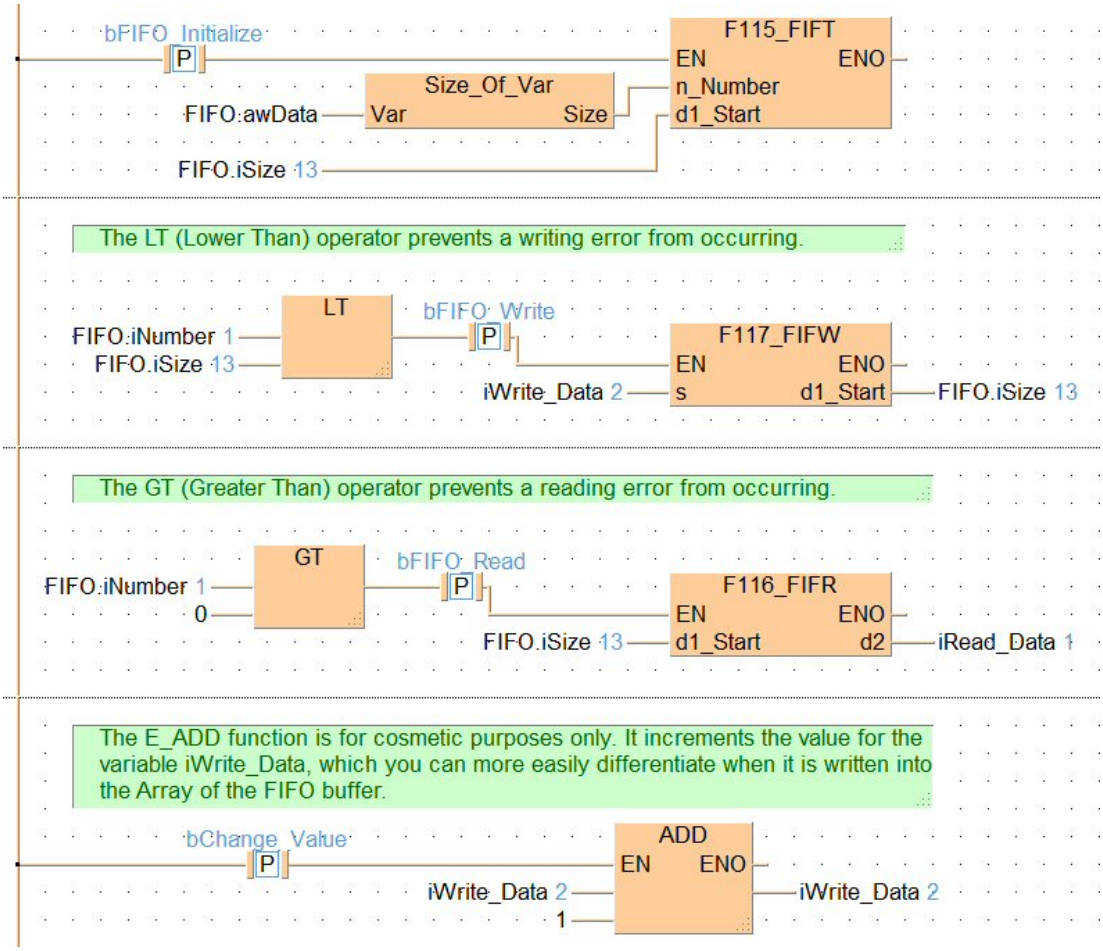
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | FIFO | FIFO_n_WORD | |
| 2 | VAR | iRead_Data | INT | 0 |
| 3 | VAR | iWrite_Data | INT | 1 |
| 4 | VAR | bFIFO_Initialize | BOOL | FALSE |
| 5 | VAR | bFIFO_Write | BOOL | FALSE |
| 6 | VAR | bFIFO_Read | BOOL | FALSE |
| 7 | VAR | bChange_Value | BOOL | FALSE |

## POU body

The example below illustrates the status of the buffer after **bFIFO_Write** has been enabled twice and **bFIFO_Read** once. When **bFIFO_Write** was activated the first time, the value 1 was written into **FIFO.awData[0]**. When **bFIFO_Read** was enabled, **iRead_Data** then read this value. When **bFIFO_Write** was enabled the second time, the writing pointer was incremented by one and the value 2 written into **FIFO.awData[1]**. see Entry Data Monitor 1

| | Identifier | Value | FP address |
|---|---|---|---|
| 1 | F115_FIFT_F116_F117 | | |
| 2 | FIFO | | |
| 3 | iSize | 13 | DT3293 |
| 4 | iNumber | 1 | DT3294 |
| 5 | wPositions | 16#0102 | DT3295 |
| 6 | awData | | |
| 7 | [0] | 16#0001 | DT3296 |
| 8 | [1] | 16#0002 | DT3297 |
| 9 | [2] | 16#0000 | DT3298 |
| 10 | [3] | 16#0000 | DT3299 |
| 11 | [4] | 16#0000 | DT3300 |
| 12 | [5] | 16#0000 | DT3301 |
| 13 | [6] | 16#0000 | DT3302 |
| 14 | [7] | 16#0000 | DT3303 |
| 15 | [8] | 16#0000 | DT3304 |
| 16 | [9] | 16#0000 | DT3305 |
| 17 | [10] | 16#0000 | DT3306 |
| 18 | [11] | 16#0000 | DT3307 |
| 19 | [12] | 16#0000 | DT3308 |
| 20 | iRead_Data | 1 | DT3309 |
| 21 | iWrite_Data | 2 | DT3310 |
| 22 | bFIFO_Initialize | TRUE | R264 |
| 23 | bFIFO_Write | TRUE | R265 |
| 24 | bFIFO_Read | TRUE | R266 |
| 25 | bChange_Value | TRUE | R267 |

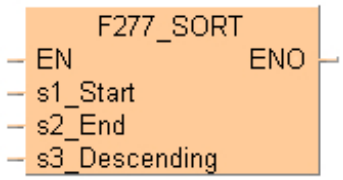
## LD body


bFIFO_Initialize
P
F115_FIFT
EN
ENO
Size_Of_Var
FIFO.awData
Var
Size
n_Number
d1_Start
FIFO.iSize 13
The LT (Lower Than) operator prevents a writing error from occurring.
LT
FIFO.iNumber 1
FIFO.iSize 13
bFIFO_Write
F117_FIFW
iWrite_Data 2
s
d1_Start
The GT (Greater Than) operator prevents a reading error from occurring.
GT
FIFO.iNumber 1
0
bFIFO_Read
F116_FIFR
d2
iRead_Data 1
The E_ADD function is for cosmetic purposes only. It increments the value for the variable iWrite_Data, which you can more easily differentiate when it is written into the Array of the FIFO buffer.
bChange_Value
ADD
1

# F277_SORT

Sort data in 16-bit data table

The function sorts values (with +/- sign) in a data table in ascending or descending order.




## Parameters

Input

**s1_Start** (INT)

Starting area of data table to be sorted

**s2_End** (INT)

Ending area of data table to be sorted

**s3_Descending** (INT)

Specifies sorting order:

- 0 = ascending
- 1 = descending

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_DATA_SORT**
- The data are sorted via bubble sort in the order specified according to the value entered at input **s3_Descending**. Since the number of word comparisons increases in proportion to the square of the number of words, the sorting process can take some time when there are a large number of words. When the value at inputs **s1_Start**=**s2_End**, no sorting takes place.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

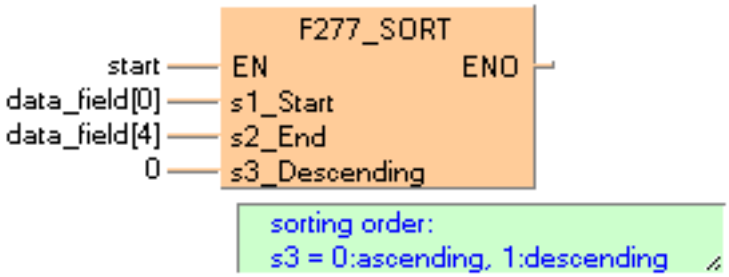
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | data_field | ARRAY [0..4] OF INT | [2,3,6,-3,1] | Arbitrarily large data field |
| 2 | VAR | | | | result: here [-3,1,2,3,6] |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

The constant 0 is specified at input s3, which means the sorting is carried out in an ascending order. However, you can declare a variable in the POU header and write it in the function in the body at input s3.

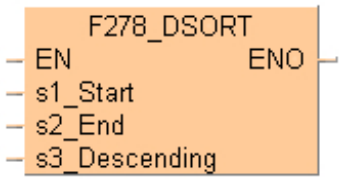
### LD body



# F278_DSORT

Sort data in 32-bit data table (in smaller or larger number order)

The function sorts values (with +/- sign) in a data table in ascending or descending order.




## Parameters

Input

**s1_Start** (DINT)

Starting area of data table to be sorted

**s2_End** (DINT)

Ending area of data table to be sorted

**s3_Descending** (INT)

Specifies sorting order:

- 0 = ascending
- 1 = descending

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_DATA_SORT**
- The data are sorted via bubble sort in the order specified according to the value entered at input **s3_Descending**. Since the number of word comparisons increases in proportion to the square of the number of words, the sorting process can take some time when there are a large number of words. When the value at inputs **s1_Start=s2_End**, no sorting takes place.
- Although this is a 32-bit instruction, the number of steps is the same as the 16-bit instruction.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**

- if **s1_Start** and **s2_End** belong to different data areas.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the address of the variable at input **s1_Start** > **s2_End**
- if **s1_Start** and **s2_End** belong to different data areas.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | data_field | ARRAY [0..4] OF DINT | [2,3,222222,-333333,1] | Arbitrarily large data field |
| 2 | VAR | sort_order | INT | 1 | 0:ascending, 1:descending |

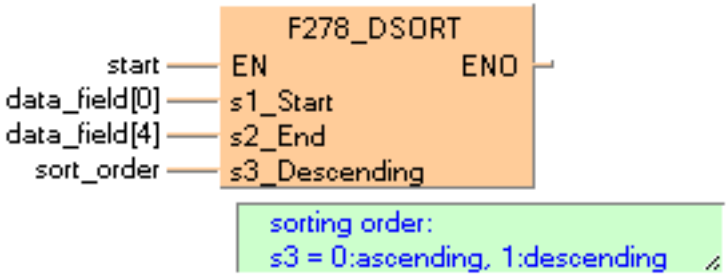
In this example, the input variable **sort_order** is declared. However, you can write a constant directly at the input contact of the function instead.

### POU body

When the variable **start** is set to TRUE, the function is carried out.

Since the variable **sort_order** is set to 1, the specified data field in sorted in descending order.

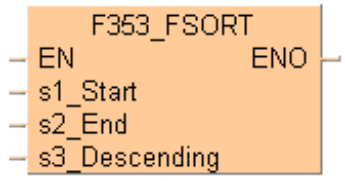
### LD body



# F353_FSORT

Sort data in real number data table (floating point data table)

The function sorts values (with +/- sign) in a data table in ascending or descending order.




## Parameters

Input

**s1_Start** (REAL)

starting area of data table to be sorted

**s2_End** (REAL)

ending area of data table to be sorted

**s3_Descending** (INT)

specifies sorting order:

- 0 = ascending
- 1 = descending

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_DATA_SORT** (page 1006)
- Input **s1_Start** specifies the starting area of the data table, and **s2_End** specifies the end. You determine the sorting order at input **s3_Descending**.
- At input **s3_Descending** you can enter the following values:

| | |
|---|---|
| 0 | ascending order, i.e. begin with the smallest value |
| 1 | descending order, i.e. begin with the largest value |

- The data are sorted via bubble sort in the order specified according to the value entered at input **s1**. Since the number of word comparisons increases in proportion to the square of the number of words, the sorting process can take some time when there are a large number of words. When the value at inputs **s1_Start**=**s2_End**, no sorting takes place.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the address of the variable at input **s1_Start** > **s2_End**
- if the address areas of the values at inputs **s1_Start** and **s2_End** are different
- if the floating point values exceed the processing range.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the address of the variable at input **s1_Start** > **s2_End**
- if the address areas of the values at inputs **s1_Start** and **s2_End** are different
- if the floating point values exceed the processing range.

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | data_field | ARRAY [0..4] OF REAL | [2.0,3.45,-6.91,5.44,1.3] | Arbitrarily large data field |
| 2 | VAR | sort_order | INT | 0 | 0:ascending, 1:descending |

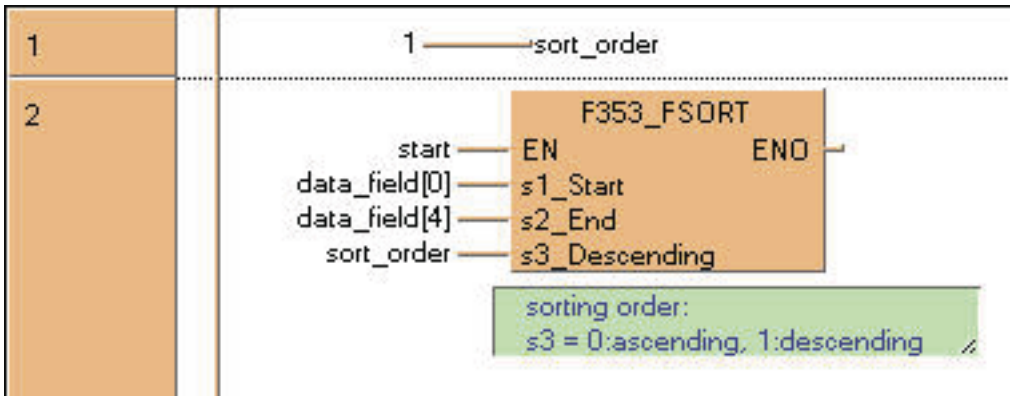
In this example, the input variable **sort_order** is declared. However, you can write a constant (e.g. 1 for a descending sorting order) directly at the input contact of the function in the body.

## POU body

The variable **sort_order** is specified as the value 1.
When the variable **start** is set to TRUE, the function is carried out.

It sorts the elements of the ARRAY **data_field** in descending order.

**LD body**

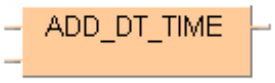

1
2
1
sort_order
F353_FSORT
start
EN
ENO
data_field[0]
s1_Start
data_field[4]
s2_End
sort_order
s3_Descending
sorting order:
s3 = 0:ascending, 1:descending

# 16 Date and time instructions

# ADD_DT_TIME

Add TIME to DATE_AND_TIME

**ADD_DT_TIME** adds the value of a variable of the data type TIME to the date and time stored in the variable of the data type DATE_AND_TIME. The result is stored in a variable of the data type DATE_AND_TIME.




## Parameters

Input

**Unnamed input** (DATE_AND_TIME)

1st input: augend

**Unnamed input** (TIME)

2nd input: addend

Output

**Unnamed output** (DATE_AND_TIME)

sum

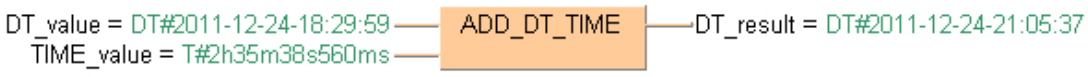
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DT_value | DATE_AND_TIME | DT#2011-12-24-18:29:59 |
| 1 | VAR | TIME_value | TIME | T#2h35m38s560ms |
| 2 | VAR | DT_result | DATE_AND_TIME | DT#2001-01-01-00:00:00 |

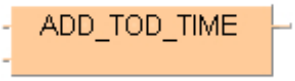
### LD body



# ADD_TOD_TIME

Add TIME to TIME_OF_DAY

**ADD_TOD_TIME** adds a variable of the data type TIME to the time of day. The result is stored in a variable of the data type TIME_OF_DAY.




## Parameters

Input

**Unnamed input** (TIME_OF_DAY)

1st input: augend

**Unnamed input** (TIME)

2nd input: addend

Output

**Unnamed output** (TIME_OF_DAY)

sum

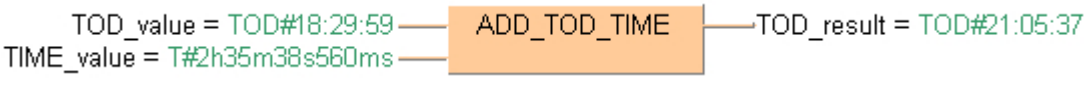
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | TOD_value | TIME_OF_DAY | TOD#18:29:59 |
| 1 | VAR | TIME_value | TIME | T#2h35m38s560ms |
| 2 | VAR | TOD_result | TIME_OF_DAY | TOD#00:00:00 |

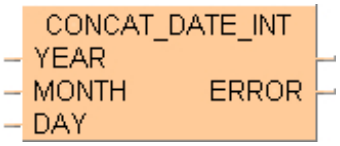
### LD body



# CONCAT_DATE_INT

Concatenate INT values to form a date

**CONCAT_DATE_INT** concatenates the INT values of year, month, and day. The result is stored in the output variable of the data type DATE. The Boolean output ERROR is set if the input values are invalid date or time values.




## Parameters

Input

**YEAR** (INT)

1st input: year

**MONTH** (INT)

2nd input: month

**DAY** (INT)

3rd input: day

Output

**VAR_OUT** (DATE)

Result

**ERROR** (BOOL)

The Boolean output ERROR is set if the input values are invalid date or time values.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

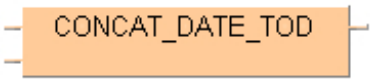
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DATE_value | DATE | D#2001-01-01 |
| 1 | VAR | YEAR_value | INT | 2011 |
| 2 | VAR | MONTH_value | INT | 12 |
| 3 | VAR | DAY_value | INT | 24 |
| 4 | VAR | ERROR | BOOL | FALSE |

**LD body**

# CONCAT_DATE_TOD

Concatenate date and time of day

**CONCAT_DATE_TOD** concatenates a value of the data type DATE with a value of the data type TIME_OF_DAY.The result is stored in the output variable of the data type DATE_AND_TIME.




## Parameters

Input

**Unnamed input** (DATE)

date

**Unnamed input** (TIME_OF_DAY)

time of day

Output

**Unnamed output** (DATE_AND_TIME)

Result

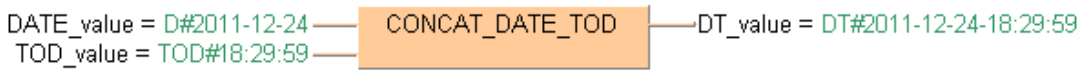
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DT_value | DATE_AND_TIME | DT#2001-01-01-00:00:00 |
| 1 | VAR | DATE_value | DATE | D#2011-12-24 |
| 2 | VAR | TOD_value | TOD | TOD#18:29:59 |

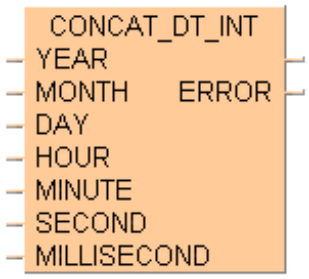
### LD body



# CONCAT_DT_INT

Concatenate INT values to form date and time

**CONCAT_DT_INT** concatenates the INT values of year, month, day, hour, minute, second, and millisecond. The result is stored in the output variable of the data type DATE_AND_TIME. The Boolean output ERROR is set if the input values are invalid date or time values.




## Parameters

Input

**YEAR** (INT)

1st input: year

**MONTH** (INT)

2nd input: month

**DAY** (INT)

3rd input: day

**HOUR** (INT)

4th input: hour

**MINUTE** (INT)

5th input: minute

**SECOND** (INT)

6th input: second

**MILLISECOND** (INT)

7th input: millisecond

Output

**VAR_OUT** (DATE_AND_TIME)

Result

**ERROR** (BOOL)

The Boolean output ERROR is set if the input values are invalid date or time values.

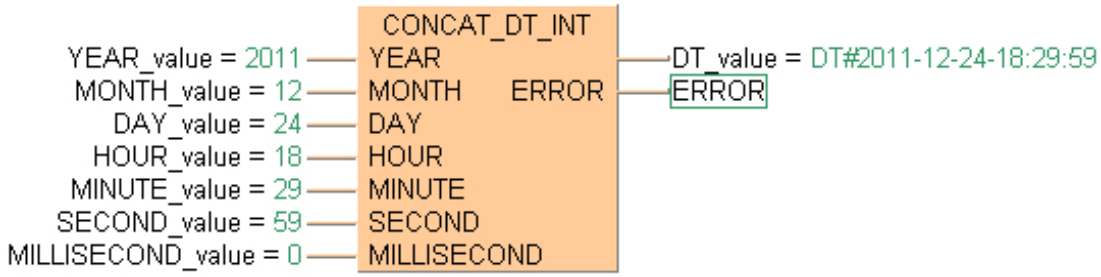
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DT_value | DATE_AND_TIME | DT#2001-01-01-00:00:00 |
| 1 | VAR | YEAR_value | INT | 2011 |
| 2 | VAR | MONTH_value | INT | 12 |
| 3 | VAR | DAY_value | INT | 24 |
| 4 | VAR | HOUR_value | INT | 18 |
| 5 | VAR | MINUTE_value | INT | 29 |
| 6 | VAR | SECOND_value | INT | 59 |
| 7 | VAR | MILLISECOND_value | INT | 0 |
| 8 | VAR | ERROR | BOOL | FALSE |

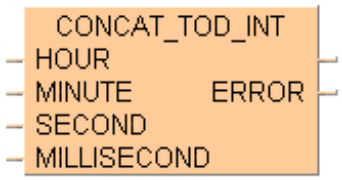
### LD body



# CONCAT_TOD_INT

Concatenate INT values to form the time of day

**CONCAT_TOD_INT** concatenates the INT values for hour, minute, second, and millisecond. The result is stored in the output variable of the data type TIME_OF_DAY. The Boolean output ERROR is set if the input values are invalid date or time values.




## Parameters

Input

**HOUR** (INT)

1st input: hour

**MINUTE** (INT)

2nd input: minute

**SECOND** (INT)

3rd input: second

**MILLISECOND** (INT)

4th input: millisecond

Output

**VAR_OUT** (TIME_OF_DAY)

Result

**ERROR** (BOOL)

The Boolean output ERROR is set if the input values are invalid date or time values.

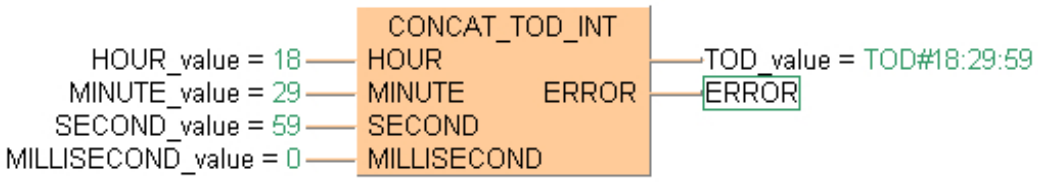
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | TOD_value | TIME_OF_DAY | TOD#00:00:00 |
| 1 | VAR | HOUR_value | INT | 18 |
| 2 | VAR | MINUTE_value | INT | 29 |
| 3 | VAR | SECOND_value | INT | 59 |
| 4 | VAR | MILLISECOND_value | INT | 0 |
| 5 | VAR | ERROR | BOOL | FALSE |

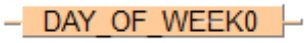
### LD body



# DAY_OF_WEEK0

Return the day of the week

**DAY_OF_WEEK0** returns the day of the week for any date as an INT. The number 0 corresponds to Sunday, 6 to Saturday.




## Parameters

Input

**Unnamed input** (DATE)

Date

Output

**Unnamed output** (WORD, INT, UINT)

0 (Sunday) – 6 (Saturday)

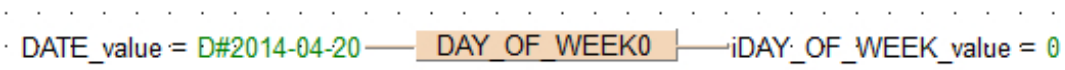
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DATE_value | DATE | D#2014-04-20 |
| 1 | VAR | iDAY_OF_WEEK_value | INT | 0 |

### LD body



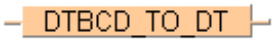

The value **iDAY_OF_WEEK_value** = 0 corresponds to Sunday.

If you need an offset of five days, e.g. 5 corresponds to Sunday:

# DTBCD_TO_DT

DTBCD to DATE_AND_TIME

**DTBCD_TO_DT** converts a value of the data type DTBCD to a value of the data type DATE_AND_TIME.




## Parameters

Input

**Unnamed input** (DTBCD)

date and time in BCD format

Output

**Unnamed output** (DATE_AND_TIME)

date and time

## Example

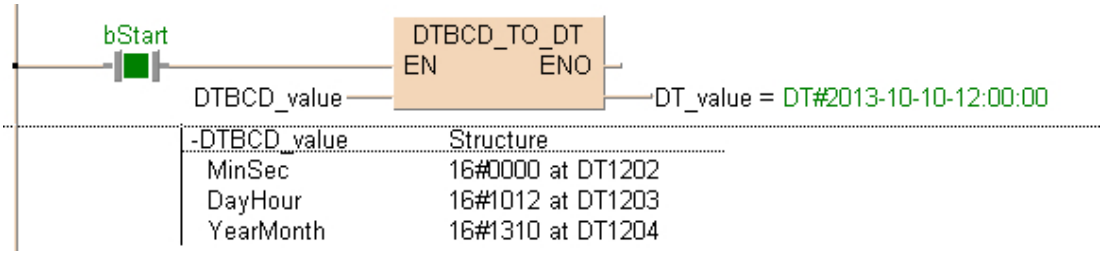
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | DT_value | DATE_AND_TIME | |
| 2 | VAR | DTBCD_value | DTBCD | MinSec := 16#0000, |

### LD body

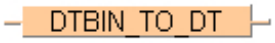
When the variable **bStart** is set to TRUE, the function is carried out.



# DTBIN_TO_DT

Date and time in binary format to DATE_AND_TIME

**DTBIN_TO_DT** converts a value of the data type DTBIN to a value of the data type DATE_AND_TIME.




## Parameters

Input

**Unnamed input** (DTBIN)

date and time in binary format

Output

**Unnamed output** (DATE_AND_TIME)

date and time

## Example

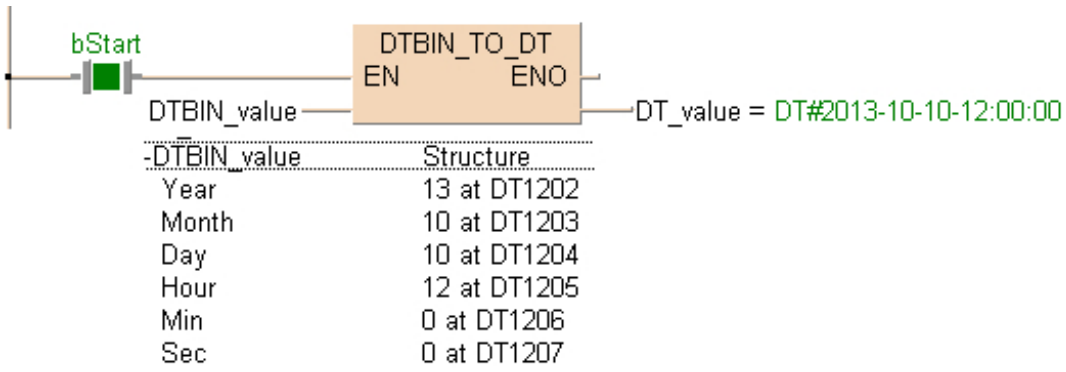
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | DT_value | DATE_AND_TIME | |
| 2 | VAR | DTBIN_value | DTBIN | Year := 13, |

### LD body

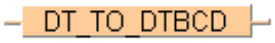
When the variable **bStart** is set to TRUE, the function is carried out.


bStart
DTBIN_TO_DT
EN
ENO
DTBIN_value
DT_value = DT#2013-10-10-12:00:00
-DTBIN_value
Structure
Year
13 at DT1202
Month
10 at DT1203
Day
10 at DT1204
Hour
12 at DT1205
Min
0 at DT1206
Sec
0 at DT1207

# DT_TO_DTBCD

DATE_AND_TIME to date and time data in BCD format

**DT_TO_DTBCD** converts a value of the data type DATE_AND_TIME to a value of the data type DTBCD.




## Parameters

Input

**Unnamed input** (DATE_AND_TIME)

date and time

Output

**Unnamed output** (DTBCD)

date and time in BCD format

## Example

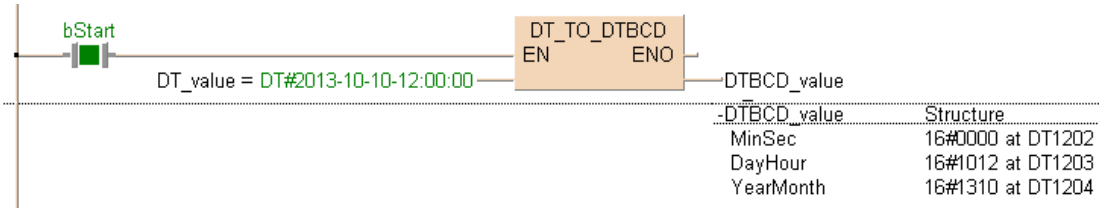
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | DT_value | DATE_AND_TIME | DT#2013-10-10-012:00:00 |
| 2 | VAR | DTBCD_value | DTBCD | |

### LD body

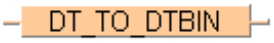
When the variable **bStart** is set to TRUE, the function is carried out.



# DT_TO_DTBIN

DATE_AND_TIME to date and time data in binary format

**DT_TO_DTBIN** converts a value of the data typeDATE_AND_TIME to a value of the data typeDTBIN.




## Parameters

Input

**Unnamed input** (DATE_AND_TIME)

date and time

Output

**Unnamed output** (DTBIN)

date and time

## Example

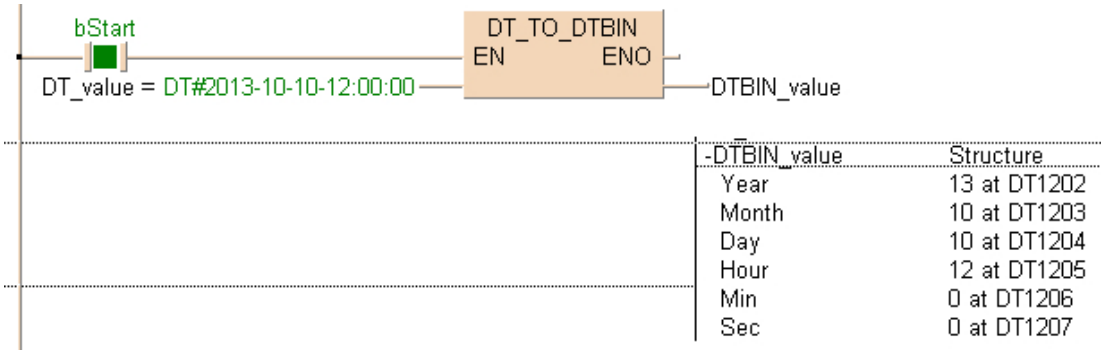
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | DT_value | DATE_AND_TIME | DT#2013-10-10-012:00:00 |
| 2 | VAR | DTBIN_value | DTBIN | |

### LD body

When the variable **bStart** is set to TRUE, the function is carried out.


bStart
DT_value = DT#2013-10-10-12:00:00
DT_TO_DTBIN
EN
ENO
DTBIN_value
-DTBIN_value
Structure
Year
13 at DT1202
Month
10 at DT1203
Day
10 at DT1204
Hour
12 at DT1205
Min
0 at DT1206
Sec
0 at DT1207

# GET_RTC_DT

Read the real-time clock

**GET_RTC_DT** reads the PLC's real-time clock value for the clock/calendar function. If the PLC has no real-time clock or if the real-time clock is not functioning, the result is an invalid date and time value.


## Parameters

Output

**VAR_OUT** (DATE_AND_TIME)

date and time

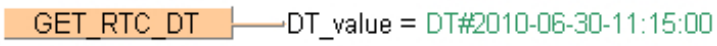
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetEdge | BOOL | FALSE |
| 1 | VAR | DT_value | DT | DT#2001-01-01-00:00:00 |

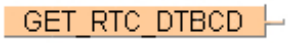
### LD body



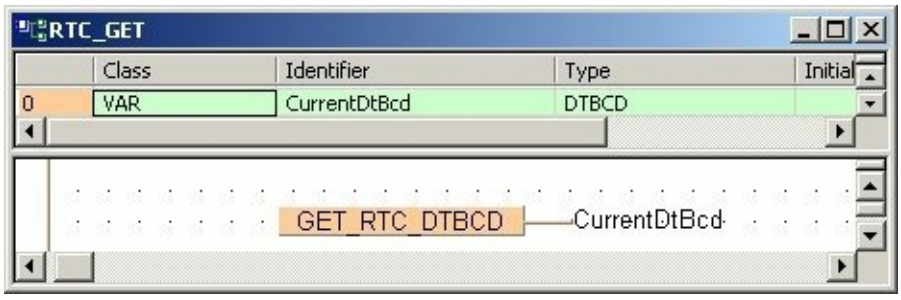
# GET_RTC_DTBCD

Read the Real-Time Clock using DTBCD

Use this PLC independent instruction to Read the Real-Time Clock using DTBCD data from the PLC. When the instruction is carried out, the values from the special data registers DT90054 to DT90056 (DT9054 to DT9056) are transferred to the data unit typeDTBCD. You can also use the system variables to set the RTC. To access special data registers and special internal flags, use the PLC-independent system variables.



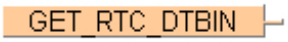

### Example



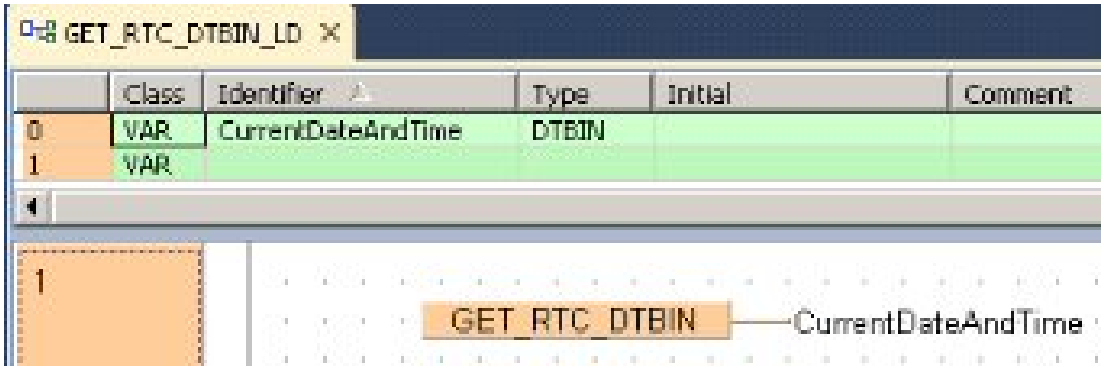
# GET_RTC_DTBIN

Read the Real-Time Clock using DTBIN

Use this instruction to read the real-time clock using DTBIN data from the PLC. When the instruction is carried out, the values from the special data registers SD50–SD56 are transferred to the data unit type DTBIN. You can also use the PLC-independent system variables to read the RTC.




**Example**

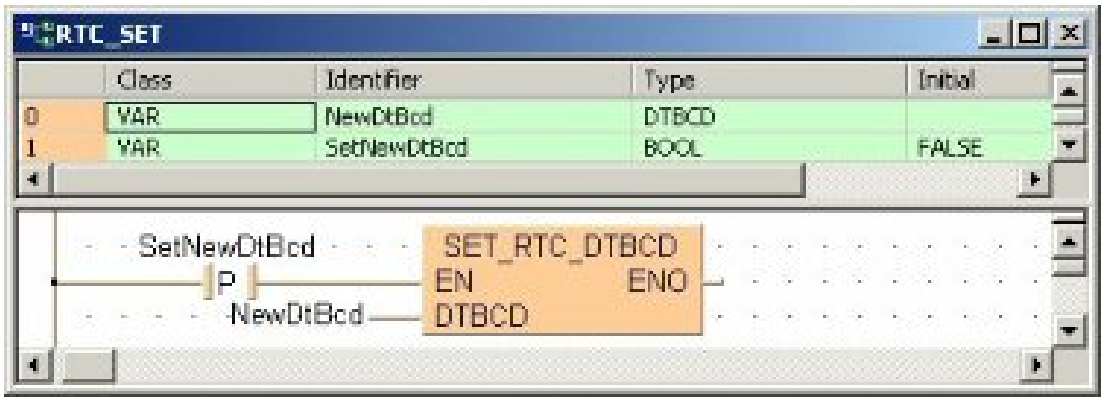


# SET_RTC_DTBCD

Set the Real-Time Clock using DTBCD

Use this PLC independent instruction to write date and time data in BCD format (DTBCD) to the real-time clock. When the variable **SetNewDtBcd** is set to TRUE, the values from the data unit type DTBCD are transferred to the special data registers DT90054 to DT90056 (DT9054 to DT9056) and the value 16#8000 is written to the special data register DT90058 (DT9058) to Set the Real-Time Clock using **DTBCD** of the PLC. You can also use the system variables to set the RTC. To access special data registers and special internal flags, use the PLC-independent system variables.


### Example

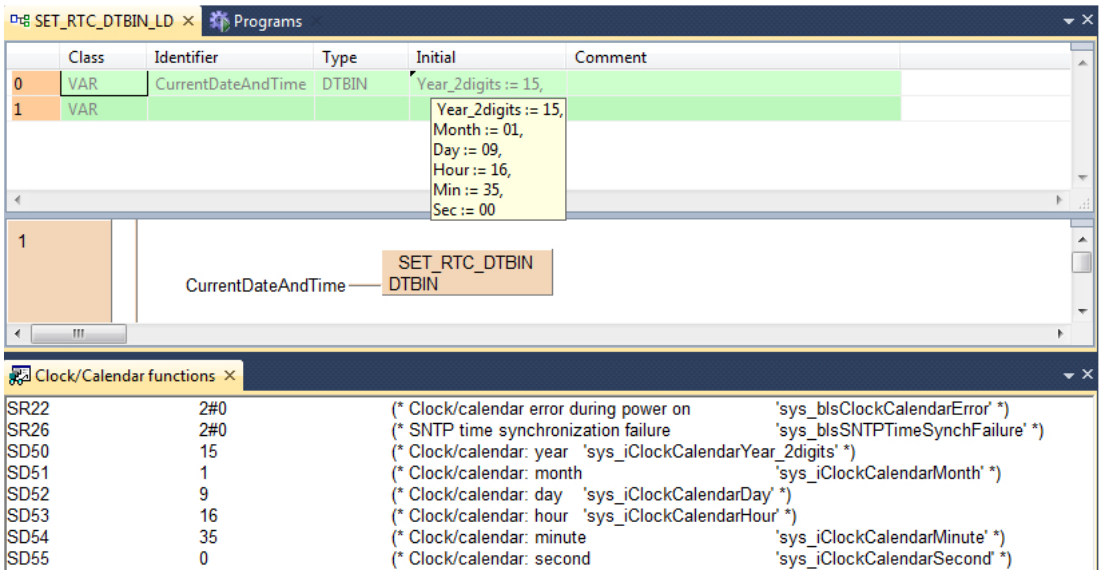


# SET_RTC_DTBIN

Set the Real-Time Clock using DTBIN

Use this instruction to write date and time data in binary formatDTBIN to the real-time clock. The values from the data unit type DTBIN are transferred to the special data registers SD50–SD55.

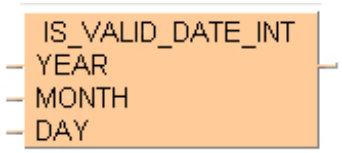

### Example



# IS_VALID_DATE_INT

Check whether DATE is valid

**IS_VALID_DATE_INT** checks whether the combination of the INT values for the year, month, and day is a valid DATE value. The Boolean output flag is set if the date is valid.




## Parameters

Input

**YEAR** (INT)

1st input: year

**MONTH** (INT)

2nd input: month

**DAY** (INT)

3rd input: day

Output

**VAR_OUT** (BOOL)

set to TRUE if the resulting date value is valid

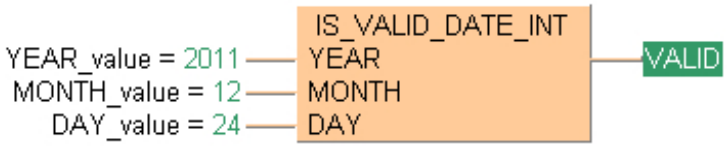
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | YEAR_value | INT | 2011 |
| 1 | VAR | MONTH_value | INT | 12 |
| 2 | VAR | DAY_value | INT | 24 |
| 3 | VAR | VALID | BOOL | FALSE |

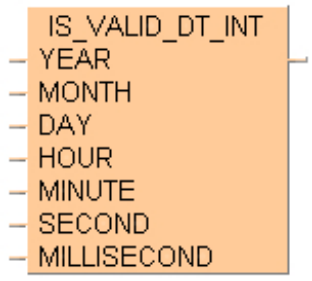
### LD body


IS_VALID_DATE_INT
YEAR_value = 2011
YEAR
MONTH_value = 12
MONTH
DAY_value = 24
DAY
VALID

# IS_VALID_DT_INT

Check whether DATE_AND_TIME is valid

**IS_VALID_DT** checks whether the combination of INT values for year, month, day, hour, minute, second, and millisecond is a valid date and time value. The Boolean output flag is set if the date and time value is valid.




## Parameters

Input

**YEAR** (INT)

1st input: year

**MONTH** (INT)

2nd input: month

**DAY** (INT)

3rd input: day

**HOUR** (INT)

4th input: hour

**MINUTE** (INT)

5th input: minute

**SECOND** (INT)

6th input: second

**MILLISECOND** (INT)

7th input: millisecond

Output

**VAR_OUT** (BOOL)

set to TRUE if the resulting date and time value is valid

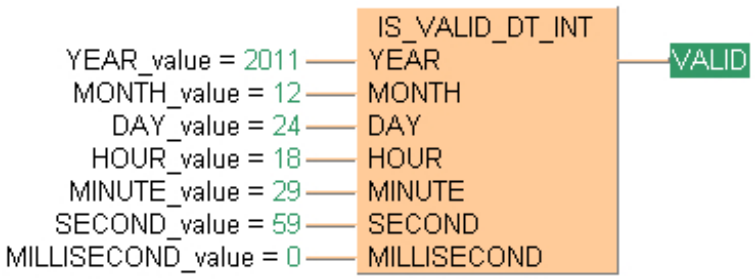
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | YEAR_value | INT | 2011 |
| 1 | VAR | MONTH_value | INT | 12 |
| 2 | VAR | DAY_value | INT | 24 |
| 3 | VAR | HOUR_value | INT | 18 |
| 4 | VAR | MINUTE_value | INT | 29 |
| 5 | VAR | SECOND_value | INT | 59 |
| 6 | VAR | MILLISECOND_value | INT | 0 |
| 7 | VAR | VALID | BOOL | FALSE |

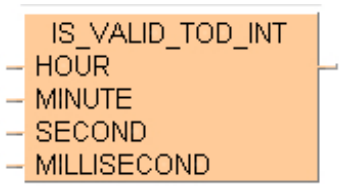
### LD body



# IS_VALID_TOD_INT

Check whether the TIME_OF_DAY is valid

**IS_VALID_TOD_INT** checks whether the combination of INT values for hour, minute, second, and millisecond is a valid time of day value. The Boolean output flag is set if the time of day value is valid.




## Parameters

Input

**HOUR** (INT)

1st input: hour

**MINUTE** (INT)

2nd input: minute

**SECOND** (INT)

3rd input: second

**MILLISECOND** (INT)

4th input: millisecond

Output

**VAR_OUT** (BOOL)

set to TRUE if the resulting time of day value is valid

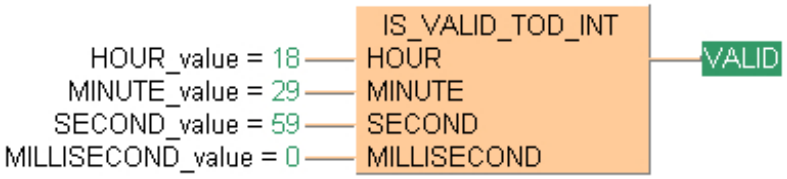
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | HOUR_value | INT | 18 |
| 1 | VAR | MINUTE_value | INT | 29 |
| 2 | VAR | SECOND_value | INT | 59 |
| 3 | VAR | MILLISECOND_value | INT | 0 |
| 4 | VAR | VALID | BOOL | FALSE |

**LD body**



# SET_RTC_DT

Set the real-time clock

**SET_RTC_DT** sets the real-time clock value in the PLC for the clock/calendar function. If the PLC has no real-time clock or if the real-time clock is not functioning, the result is an invalid date and time value.

From version 6.42 onwards, this function implicitely sets the day of week with the function DAY_OF_WEEK0, i.e. 0 corresponds to Sunday, 6 to Saturday.


## Parameters

Input

**IN** (DATE_AND_TIME)

date and time

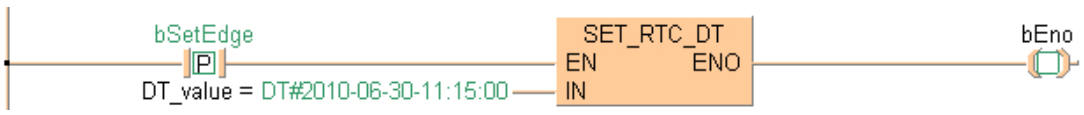
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetEdge | BOOL | FALSE |
| 1 | VAR | DT_value | DT | DT#2010-06-30-11:15:00 |
| 2 | VAR | bEno | BOOL | FALSE |

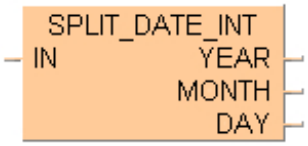
### LD body



# SPLIT_DATE_INT

Split DATE into INT values

**SPLIT_DATE_INT** splits a value of the data type DATE into INT values for year, month, and day.




## Parameters

Input

**IN** (DATE)

date

Output

**YEAR** (INT)

year

**MONTH** (INT)

month

**DAY** (INT)

day

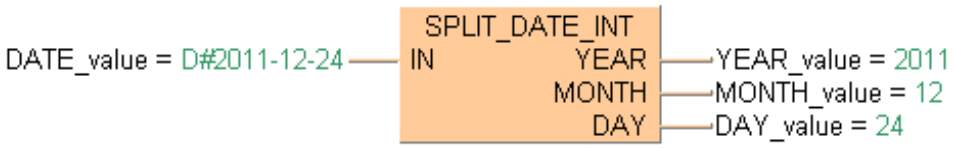
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DATE_value | DATE | D#2011-12-24 |
| 1 | VAR | YEAR_value | INT | 0 |
| 2 | VAR | MONTH_value | INT | 0 |
| 3 | VAR | DAY_value | INT | 0 |

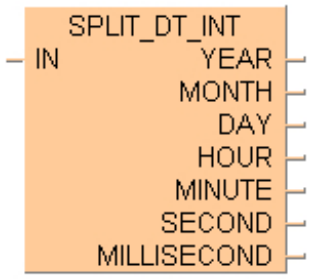
**LD body**


DATE_value = D#2011-12-24
SPLIT_DATE_INT
IN
YEAR
MONTH
DAY
YEAR_value = 2011
MONTH_value = 12
DAY_value = 24

# SPLIT_DT_INT

Split DATE_AND_TIME into INT values

**SPLIT_DT_INT** splits a value of the data type DATE_AND_TIME into INT values for year, month, day, hour, minute, second, and millisecond.




## Parameters

Input

**IN** (DATE_AND_TIME)

date and time

Output

**YEAR** (INT)

year

**MONTH** (INT)

month

**DAY** (INT)

day

**HOUR** (INT)

hour

**MINUTE** (INT)

minute

**SECOND** (INT)

second

**MILLISECOND** (INT)

millisecond

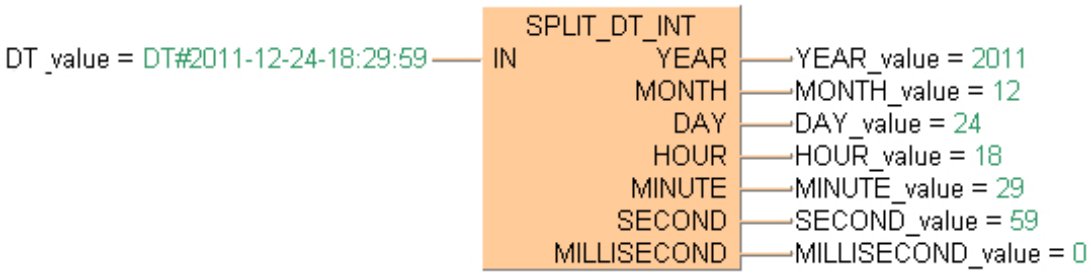
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DT_value | DATE_AND_TIME | DT#2011-12-24-18:29 |
| 1 | VAR | YEAR_value | INT | 0 |
| 2 | VAR | MONTH_value | INT | 0 |
| 3 | VAR | DAY_value | INT | 0 |
| 4 | VAR | HOUR_value | INT | 0 |
| 5 | VAR | MINUTE_value | INT | 0 |
| 6 | VAR | SECOND_value | INT | 0 |
| 7 | VAR | MILLISECOND_value | INT | 0 |

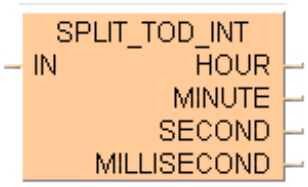
### LD body



# SPLIT_TOD_INT

Split TIME_OF_DAY into INT values

**SPLIT_TOD_INT** splits a value of the data type TIME_OF_DAY into INT values for hour, minute, second, and millisecond.




## Parameters

Input

**IN** (TIME_OF_DAY)

time of day

Output

**HOUR** (INT)

hour

**MINUTE** (INT)

minute

**SECOND** (INT)

second

**MILLISECOND** (INT)

millisecond

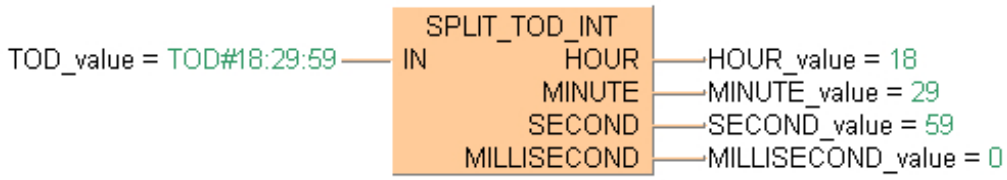
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | TOD_value | TIME_OF_DAY | TOD#18:29:59 |
| 1 | VAR | HOUR_value | INT | 0 |
| 2 | VAR | MINUTE_value | INT | 0 |
| 3 | VAR | SECOND_value | INT | 0 |
| 4 | VAR | MILLISECOND_value | INT | 0 |

**LD body**



# SUB_DATE_DATE

Subtract DATE from DATE

**SUB_DATE_DATE** subtracts a value of the data type DATE from another DATE value. The result is stored in the output variable of the data type TIME.


## Parameters

Input

**Unnamed input** (DATE)

1st input: minuend

**Unnamed input** (DATE)

2nd input: subtrahend

Output

**Unnamed output** (TIME)

Result

## Remarks

The TIME result is only valid if the difference between the minuend and subtrahend is smaller than or equal to the maximum TIME duration allowed. Otherwise an overflow of the TIME result variable occurs and the CARRY flag is set.

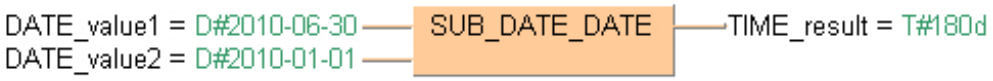
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DATE_value1 | DATE | D#2010-06-30 |
| 1 | VAR | DATE_value2 | DATE | D#2010-01-01 |
| 2 | VAR | TIME_result | TIME | T#0s |

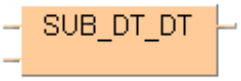
**LD body**


DATE_value1 = D#2010-06-30
DATE_value2 = D#2010-01-01
SUB_DATE_DATE
TIME_result = T#180d

# SUB_DT_DT

Subtract DATE_AND_TIME from DATE_AND_TIME

**SUB_DT_DT** subtracts a value of the data type DATE_AND_TIME from another DATE_AND_TIME value. The result is stored in the output variable of the data type TIME.




## Parameters

Input

**Unnamed input** (DATE_AND_TIME)

1st input: minuend

**Unnamed input** (DATE_AND_TIME)

2nd input: subtrahend

Output

**Unnamed output** (TIME)

Result

## Remarks

The TIME result is only valid if the difference between the minuend and subtrahend is smaller than or equal to the maximum TIME duration allowed. Otherwise an overflow of the TIME result variable occurs and the CARRY flag is set.

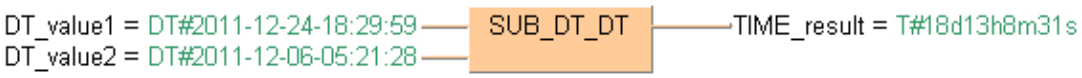
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DT_value1 | DATE_AND_TIME | DT#2011-12-24-18:29:59 |
| 1 | VAR | DT_value2 | DATE_AND_TIME | DT#2011-12-06-05:21:28 |
| 2 | VAR | TIME_result | TIME | T#0s |

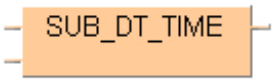
### LD body


DT_value1 = DT#2011-12-24-18:29:59
DT_value2 = DT#2011-12-06-05:21:28
SUB_DT_DT
TIME_result = T#18d13h8m31s

# SUB_DT_TIME

Subtract TIME from DATE_AND_TIME

**SUB_DT_TIME** subtracts a value of the data type TIME from a value of the data type DATE_AND_TIME. The result is stored in the output variable of the data type DATE_AND_TIME.




## Parameters

Input

**Unnamed input** (DATE_AND_TIME)

1st input: minuend

**Unnamed input** (TIME)

2nd input: subtrahend

Output

**Unnamed output** (DATE_AND_TIME)

Result

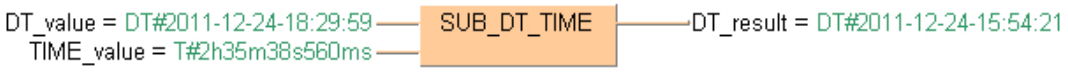
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | DT_value | DATE_AND_TIME | DT#2011-12-24-18:29:59 |
| 1 | VAR | TIME_value | TIME | T#2h35m38s560ms |
| 2 | VAR | DT_result | DATE_AND_TIME | DT#2001-01-01-00:00:00 |

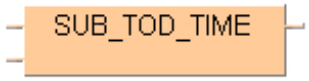
### LD body



# SUB_TOD_TIME

Subtract TIME from TIME_OF_DAY

**SUB_TOD_TIME** subtracts a TIME value from a value of the data type TIME_OF_DAY. The result is stored in the output variable of the data type TIME_OF_DAY.




## Parameters

Input

**Unnamed input** (TIME_OF_DAY)

1st input: minuend

**Unnamed input** (TIME)

2nd input: subtrahend

Output

**Unnamed output** (TIME_OF_DAY)

Result

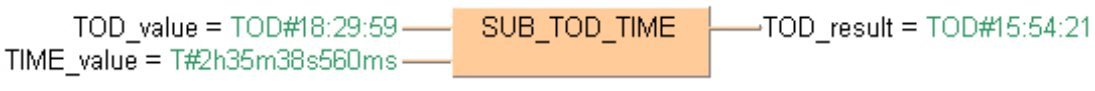
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | TOD_value | TIME_OF_DAY | TOD#18:29:59 |
| 1 | VAR | TIME_value | TIME | T#2h35m38s560ms |
| 2 | VAR | TOD_result | TIME_OF_DAY | TOD#00:00:00 |

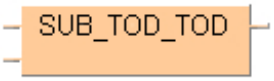
### LD body



# SUB_TOD_TOD

Subtract TIME_OF_DAY from TIME_OF_DAY

**SUB_TOD_TOD** subtracts a value of the data typeTIME_OF_DAY from another TIME_OF_DAY value. The result is stored in the output variable of the data type TIME.




## Parameters

Input

**Unnamed input** (TIME_OF_DAY)

1st input: minuend

**Unnamed input** (TIME_OF_DAY)

2nd input: subtrahend

Output

**Unnamed output** (TIME)

Result

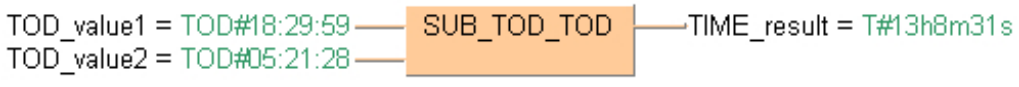
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | TOD_value1 | TIME_OF_DAY | TOD#18:29:59 |
| 1 | VAR | TOD_value2 | TIME_OF_DAY | TOD#05:21:28 |
| 2 | VAR | TIME_result | TIME | T#0s |

### LD body



## 16.29 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

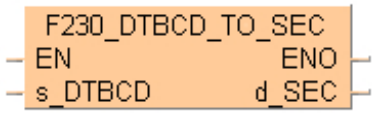
Advantages of IEC instructions

Advantages of FP instructions

# F230_DTBCD_TO_SEC

Time data conversion into seconds

This function converts time data (date and time) into the number of seconds. It calculates the time span between the specified time date and 01/01/2001 at 00:00 hours. The time data is specified in the DUT "DTBCD".




## Parameters

Input

**s_DTBCD** (DTBCD)

Area in which the input time data is stored

Output

**d_SEC** (DWORD, DINT, UDINT, DATE, TOD, DT)

Area in which the converted second information is stored (32 bits)

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the specified address using the index modifier exceeds a limit.
- if values other than BCD are specified for **s_DTBCD**.
- if the value which exceeds the range in the time data of 's' is specified.

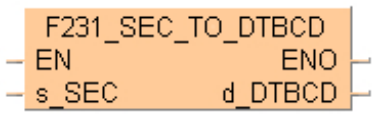
**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the specified address using the index modifier exceeds a limit.
- if values other than BCD are specified for **s_DTBCD**.
- if the value which exceeds the range in the time data of 's' is specified.

# F231_SEC_TO_DTBCD

Conversion of seconds into time data

This function converts a specified number of seconds into date and time. The time data is calculated from 01/01/2001 at 00:00 hours.




## Parameters

Input

**s_SEC** (DWORD, DINT, UDINT, DATE, TOD, DT)

Area in which the number of seconds are stored (32 bits)

Output

**d_DTBCD** (DTBCD)

Head area in which time data is stored

## Error flags

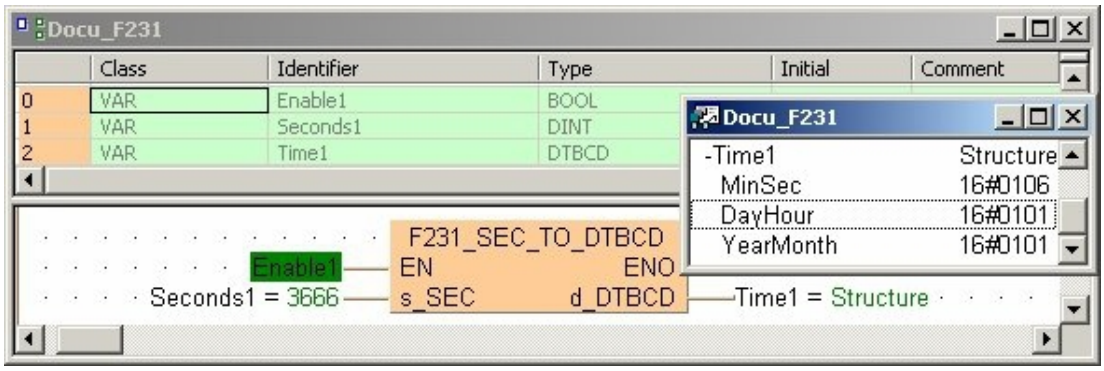
**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the specified address using the index modifier exceeds a limit.
- if the number of seconds **s_SEC** >= 16#BC191380 (valid until 31 Dec. 2100 23:59:59).
- if the data memory of **d_DTBCD** exceeds the area.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the specified address using the index modifier exceeds a limit.
- if the number of seconds **s_SEC** >= 16#BC191380 (valid until 31 Dec. 2100 23:59:59).
- if the data memory of **d_DTBCD** exceeds the area.

### Example

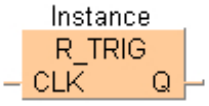

Docu_F231
Class
Identifier
Type
Initial
Comment
0
VAR
Enable1
BOOL
1
VAR
Seconds1
DINT
2
VAR
Time1
DTBCD
Docu_F231
-Time1
Structure
MinSec
16#0106
DayHour
16#0101
YearMonth
16#0101
F231_SEC_TO_DTBCD
Enable1
EN
ENO
Seconds1 = 3666
s_SEC
d_DTBCD
Time1 = Structure

# 17 Edge detection instructions

# R_TRIG

Rising edge trigger

The function block **R_TRIG** (rising edge trigger) allows you to recognize a rising edge at an input.




## Parameters

Input

**CLK** (BOOL)

signal input
detects rising edge for clock

Output

**Q** (BOOL)

signal output
is set for each rising edge at the signal input **CLK** (clock)

## Remarks

The output **Q** of a function block **R_TRIG** remains set for a complete PLC cycle after the occurrence of a rising edge (status change FALSE -> TRUE) at the **CLK** input and is then reset in the following cycle.

## Example

### POU header

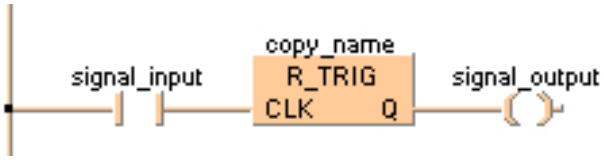
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | copy_name | R_TRIG | |
| 1 | VAR | signal_input | BOOL | FALSE |
| 2 | VAR | signal_output | BOOL | FALSE |

### POU body

**Signal_output** will be set, if a rising edge is detected at **signal_input**.

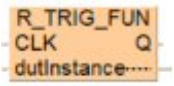
### LD body



# R_TRIG_FUN

Rising edge trigger

This is a user-defined function from a system function block. **R_TRIG_FUN** (rising edge trigger) allows you to recognize a rising edge at an input.




## Parameters

Input

**CLK** (BOOL)

signal input
detects rising edge for clock

Input/output

**dutInstance**(R_TRIG_FUN_INSTANCE_DUT)

Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

Output

**Q** (BOOL)

signal output
is set for each rising edge at the signal input **CLK** (clock)

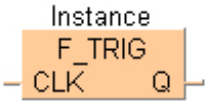
## Remarks

The output **Q** of a function **R_TRIG_FUN** remains set for a complete PLC cycle after the occurrence of a rising edge (status change FALSE -> TRUE) at the **CLK** input and is then reset in the following cycle.

# F_TRIG

Detecting a falling edge

The function block **F_TRIG** (falling edge trigger) allows you to recognize a falling edge at an input.




## Parameters

Input

**CLK** (BOOL)

signal input
detects falling edge for clock

Output

**Q** (BOOL)

signal output
is set for each falling edge at the signal input **CLK** (clock)

## Remarks

The output **Q** of a function block **F_TRIG** remains set for a complete PLC cycle after the occurrence of a falling edge (status change TRUE -> FALSE) at the **CLK** input and is then reset in the following cycle.

## Example

### POU header

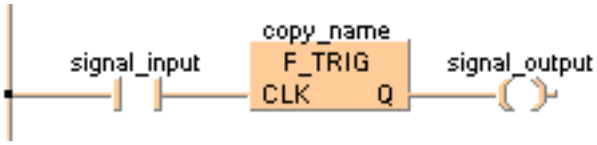
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | copy_name | F_TRIG | |
| 1 | VAR | signal_input | BOOL | FALSE |
| 2 | VAR | signal_output | BOOL | FALSE |

### POU body

**Signal_output** will be set, if a falling edge is detected at **signal_input**.

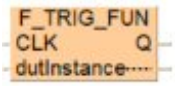
### LD body



# F_TRIG_FUN

Detecting a falling edge

This is a user-defined function from a system function block. **F_TRIG_FUN** (falling edge trigger) allows you to recognize a falling edge at an input.




## Parameters

Input

**CLK** (BOOL)

signal input
detects falling edge for clock

Input/output

**dutInstance**(F_TRIG_FUN_INSTANCE_DUT)

Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

Output

**Q** (BOOL)

signal output
is set for each falling edge at the signal input **CLK** (clock)

## Remarks

The output **Q** of a function **F_TRIG_FUN** remains set for a complete PLC cycle after the occurrence of a falling edge (status change TRUE -> FALSE) at the **CLK** input and is then reset in the following cycle.

## 17.5 FP instructions

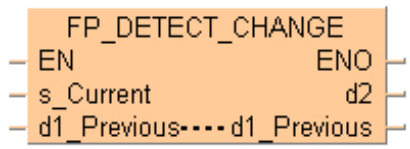
Tip

Advantages of FP instructions

# FP_DETECT_CHANGE

Detect changes in data

This FP instruction detects changes in a value at input **s_Current** by comparing it with its former value that is stored at input/output **d1_Previous**. If the new input value at **s_Current** is not equal to the former value, the function assigns the new value to the input/output **d1_Previous**. To signal the change, the system output **d2** is set to TRUE.




## Parameters

Input

**s_Current** (INT, DINT, UINT, UDINT, REAL, LREAL)

Data area for detecting data changes

Input/output

**d1_Previous** (INT, DINT, UINT, UDINT, REAL, LREAL)

Data area where the data from the previous execution is stored.

Output

**d2** (BOOL)

Set to TRUE if a change is detected

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

### POU header

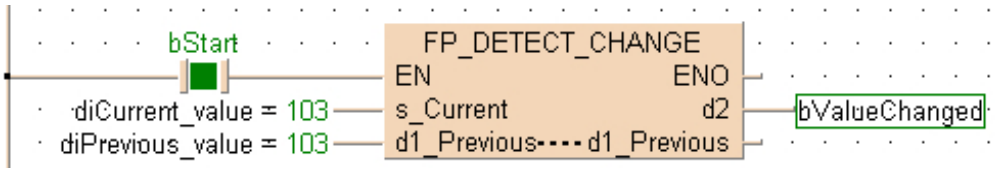
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | diCurrent_value | DINT | 101 | value whose status |
| 2 | VAR | diPrevious_value | DINT | 0 | dummy value for storing |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

### LD body



## 17.6 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

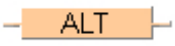
Advantages of IEC instructions

Advantages of FP instructions

# ALT

Alternative out

The function inverts the output condition each time the rising edge of the input signal is detected.




## Parameters

Input

**Unnamed input** (BOOL)

input signal

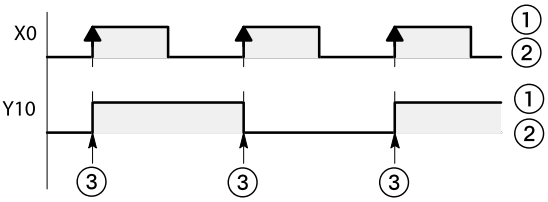
Output

**Unnamed output** (BOOL)

inverted output signal

## Remarks

When the mode is changed from “PROG” to “RUN” or the power is turned on in “RUN” mode while the input signal is TRUE, a rising edge will not be detected for the first scan.




(1) Invert

Be careful when programming with commands that effect the order in which a program is carried out, e.g. jump or loop instructions within a sequential function chart or a function block. The order of the instructions might change depending on the time when the instruction is carried out or the input value. Specific basic **JUMP** and **LOOP** instructions are:

- **MC** to **MCE** instruction
- **JP** to **LBL** instruction
- **F19_SJP** to **LBL** instruction
- **LOOP** to **LBL** instruction

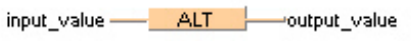
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | input_value | BOOL | FALSE |
| 1 | VAR | output_value | BOOL | FALSE |

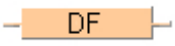
### LD body



# DF

Rising edge differential

**DF** is a rising edge differential instruction. The **DF** instruction executes and turns ON output **o** for a singular scan duration if the trigger **i** changes from an OFF to an ON state.




## Parameters

Input

**Unnamed input** (BOOL)

Output

**Unnamed output** (BOOL)

## Remarks

Be careful when programming with commands that effect the order in which a program is carried out, e.g. jump or loop instructions within a sequential function chart or a function block. The order of the instructions might change depending on the time when the instruction is carried out or the input value. Specific basic **JUMP** and **LOOP** instructions are: **F19_SJP**

## Example

## POU header

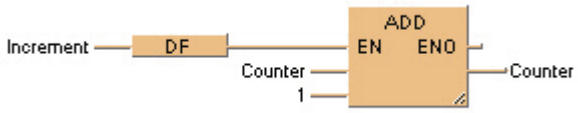
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Increment | BOOL | FALSE |
| 1 | VAR | Counter | INT | 0 |

## POU body

Each rising edge at the input **Increment** increments the counter.

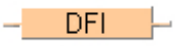
### LD body



# DFI

Rising edge differential (initial execution type)

When a rising edge of the input signal (input **i**) is detected, this function changes the status of the output signal (output **o**) to TRUE for the duration of the scan.




## Parameters

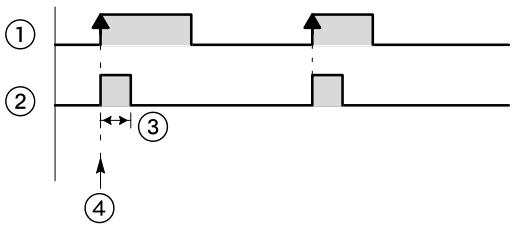
Input

**Unnamed input** (BOOL)

Output

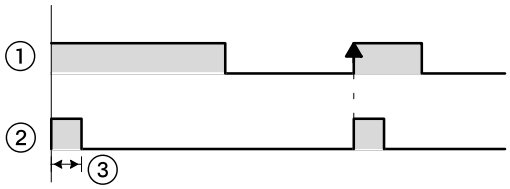
**Unnamed output** (BOOL)

## Remarks




(1) Input signal
(2) Output signal
(3) One scan
(4) Leading edge

Detection of the input signal's rising edge is also assured at the first scan.




(1) Input signal
(2) Output signal
(3) One scan

You may use an unlimited number of **DFI** functions.

If the input signal = TRUE already when the system is turned on and this signal should not be interpreted as the first rising edge, the **DF** function must be used instead.

Be careful when programming with commands that effect the order in which a program is carried out, e.g. jump or loop instructions within a sequential function chart or a function block. The order of the instructions might change depending on the time when the instruction is carried out or the input value. Specific basic **JUMP** and **LOOP** instructions are:

- **MC** to **MCE**
- **JP** to **LBL**
- **F19_SJP** to **LBL**
- **LOOP** to **LBL**

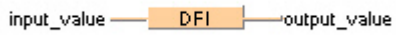
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | input_value | BOOL | FALSE |
| 1 | VAR | output_value | BOOL | FALSE |

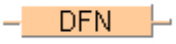
### LD body



# DFN

Falling edge differential

The **DFN** instruction executes and turns ON output **o** for a single scan duration if the trigger **i** changes from an ON to an OFF state.




## Parameters

Input

**Unnamed input** (BOOL)

Output

**Unnamed output** (BOOL)

## Remarks

Be careful when programming with commands that effect the order in which a program is carried out, e.g. jump or loop instructions within a sequential function chart or a function block. The order of the instructions might change depending on the time when the instruction is carried out or the input value. Specific basic **JUMP** and **LOOP** instructions are:

## Example

### POU header

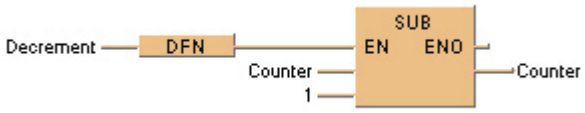
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Decrement | BOOL | FALSE |
| 1 | VAR | Counter | INT | 0 |

### POU body

Each falling edge at the input **Decrement** decrements the couter.

### LD body



# 18 FP-e display instructions

# F180_SCR

Screen display instruction

This instruction sets up the screen display in the normal mode (N) and switch mode (S) of the FP-e unit.

## Remarks

- Special register "DT9***" cannot be specified for the lower section display data "**s4**."
- This instruction cannot be used in an interrupt program.

## Parameters

Input

**s1** (WORD, INT, UINT)

Specifies "s1" registration screen

**s2** (ARRAY [0..2] OF INT, WORD)

Specifies the head of the screen display control data (3 words).

**s3** (WORD, INT, UINT)

Specifies the data displayed in the upper section.

**s4** (WORD, INT, UINT)

Specifies the data displayed in the lower section.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- the value for **s1** or **s2** is outside of the range specified.

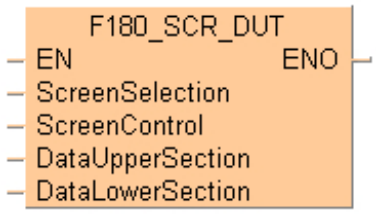
**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the value for **s1** or **s2** is outside of the range specified.

# F180_SCR_DUT

Configuring the display of the FP-e

This instruction allows you to configure the screen display of the FP-e for N mode (normal mode) and S mode (switch mode).




## Parameters

Input

**ScreenSelection** (WORD, INT, UINT)

Display mode

0: N mode 1st screen

1: N mode 2nd screen

2: S mode 1st screen

3: S mode 2nd screen

**ScreenControl** (F180_DUT)

Data unit type for the control data of the screen display.

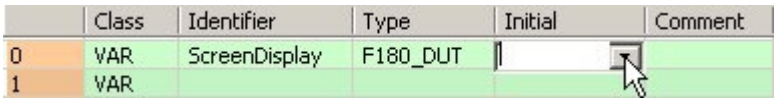
**DataUpperSection** (WORD, INT, UINT)

Value in the upper display area

**DataLowerSection** (WORD, INT, UINT)

Value in the lower display area

## Remarks




Using a convenient dialog, the control code for the screen display is configured.

1. Assigning a DUT
2. Select F180_DUT in the header of the declaration under "Type"
3. Click  in the “Initial” field

The configuration dialog opens.

4. Make desired settings
5. [OK]

Note

- You cannot enter the special data register “DT9***” for the lower display area.
- You cannot use this instruction in an interrupt program.

### Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if when the area defined by index modifiers is greater than the area allowed
- if the value for **ScreenSelection** or **ScreenControl** is invalid

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if when the area defined by index modifiers is greater than the area allowed
- if the value for **ScreenSelection** or **ScreenControl** is invalid

## Example

### Global variables

The following variables must be declared in the global variable list:

Global Variables

| | Class | Identifier | FP Address | IEC Address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | ElapsedValue0 | EV0 | %MW4.0 | INT | 88 |
| 1 | VAR_GLOBAL | ElapsedValue1 | EV1 | %MW4.1 | INT | 88 |
| 2 | VAR_GLOBAL | SetValue0 | SV0 | %MW3.0 | INT | 100 |
| 3 | VAR_GLOBAL | SetValue1 | SV1 | %MW3.1 | INT | 200 |

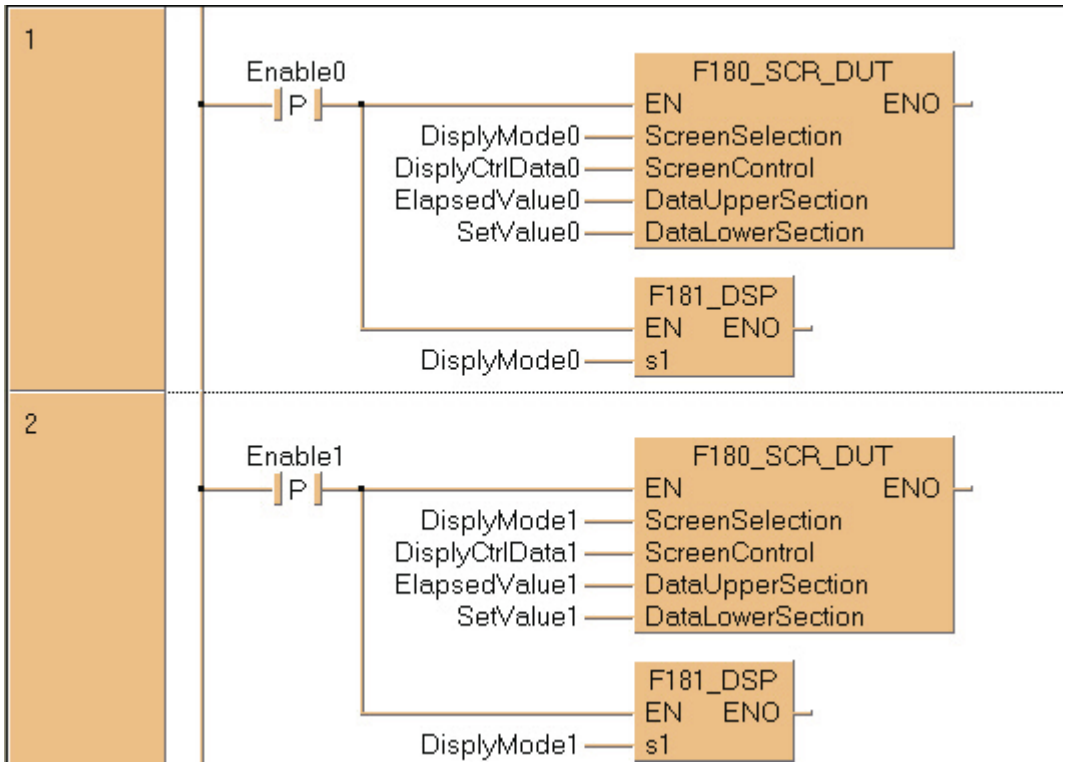
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | ElapsedValue0 | INT | 88 |
| 1 | VAR_EXTERNAL | ElapsedValue1 | INT | 88 |
| 2 | VAR_EXTERNAL | SetValue0 | INT | 100 |
| 3 | VAR_EXTERNAL | SetValue1 | INT | 200 |
| 4 | VAR | DisplayCtrlData0 | F180_DUT | ScreenControl := 16#83 |
| 5 | VAR | DisplayCtrlData1 | F180_DUT | ScreenControl := 16#83 |
| 6 | VAR | DisplayMode0 | INT | 0 |
| 7 | VAR | DisplayMode1 | INT | 1 |
| 8 | VAR | Enable0 | BOOL | FALSE |
| 9 | VAR | Enable1 | BOOL | FALSE |

When the variable **Enable0** is set to TRUE, the function is executed and the FP-e is switched to N mode, 1st screen. **ElapsedValue0** and **SetValue0** are displayed in the upper and lower sections in red and orange. When the variable **Enable1** is set to TRUE, the function is executed and the FP-e is switched to N mode, 2nd screen. **ElapsedValue1** and **SetValue1** are displayed in the upper and lower sections in red and green. The monitor value icon is activated for both LD bodies. Use the instruction **F181_DSP** to change the display of the FP-e.

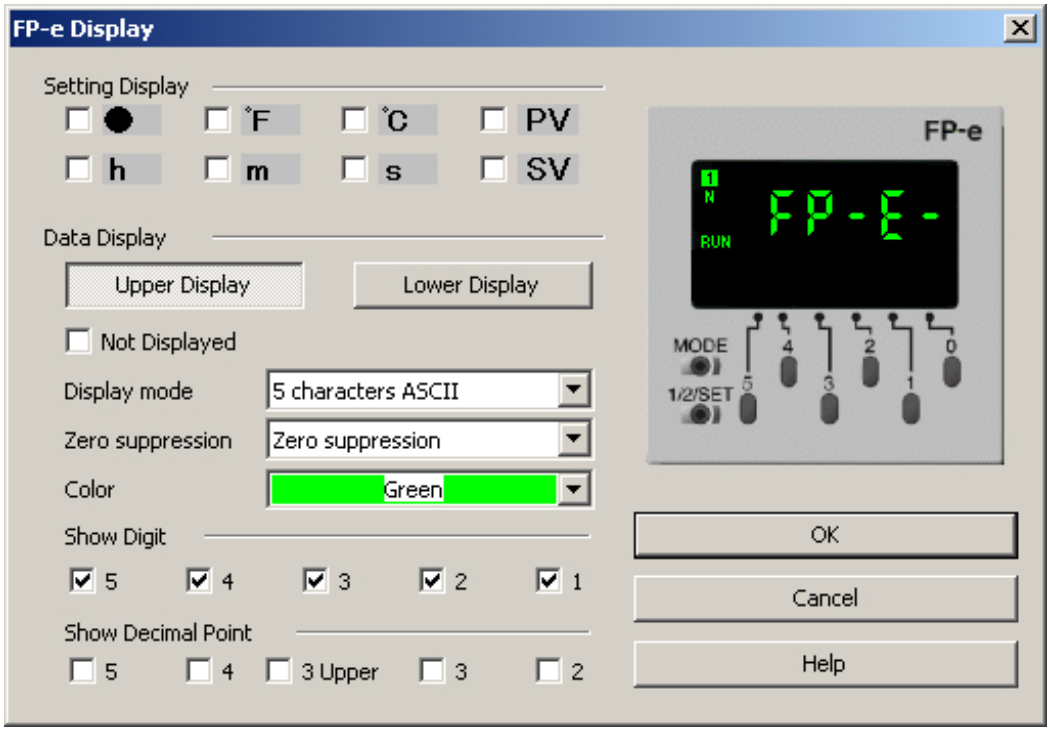
## LD body



| DisplayMode0 | DisplayMode1 |
|---|---|
| DisplayControlData0<br>ScreenControl := 16#83,<br>UpperDisplayControl := 16#4000,<br>LowerDisplayControl := 16#2000 | DisplayControlData1<br>ScreenControl := 16#83,<br>UpperDisplayControl := 16#4000,<br>LowerDisplayControl := 16#6000 |
|  |  |

## FP-e screen display

Use this dialog to configure the screen display of the FP-e easily. You can check the result of the configuration directly with the display in the dialog. You can find the meaning of the individual settings in the configuration of the screen display with control data **s2**.



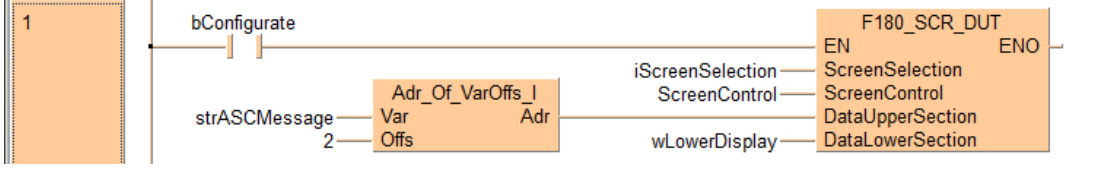
## Examples of control register for F180

| DUT element | Settings |
|---|---|
| ScreenControl | "0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1" = 16#83<br>1 SV PV<br>(1) Upper/lower section display |
| UpperDisplayControl | "0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0" = 16#4000<br>1 2 3 4<br>(1) Zero suppression<br>(2) Red<br>(3) All digits<br>(4) Decimal point: not displayed |
| LowerDisplayControl | "0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0" = 16#6000<br>1 2 3 4<br>(1) Zero suppression<br>(2) Orange<br>(3) All digits<br>(4) Decimal point: not displayed |

## Display of ASCII code with F180

All characters to be displayed are stored in the variable **strASCMessage**.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bConfigurate | BOOL | FALSE | Configurate the display |
| 1 | VAR | wLowerDisplay | WORD | 0 | Value to display in lower screen |
| 2 | VAR | iScreenSelection | INT | 0 | Select the screen |
| 3 | VAR | strASCMessage | STRING | 'abcd' | Message to display in upper screen |
| 4 | VAR | | | | |

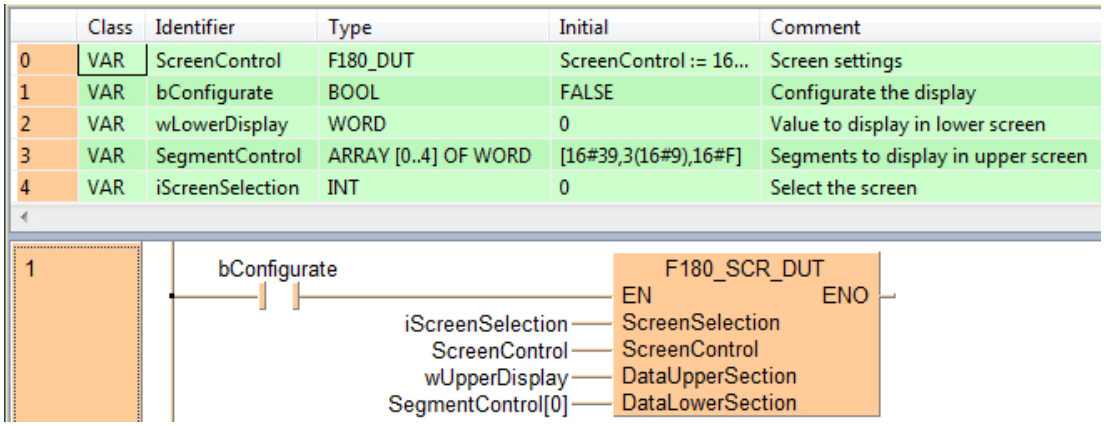


Use the instruction Adr_Of_VarOffs_I with offset 2 to access the starting position of the ASCII code within the string.

The ASCII code is displayed as follows:

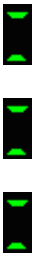

## Display of 7-segment data with F180

The segment data to be displayed is stored in the ARRAY **SegmentControl**.

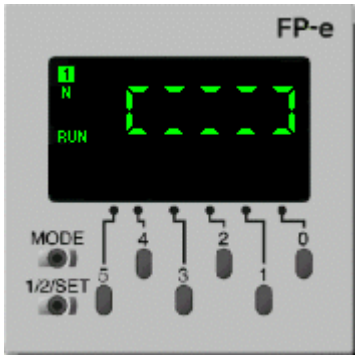


| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | ScreenControl | F180_DUT | ScreenControl := 16... | Screen settings |
| 1 | VAR | bConfigurate | BOOL | FALSE | Configurate the display |
| 2 | VAR | wLowerDisplay | WORD | 0 | Value to display in lower screen |
| 3 | VAR | SegmentControl | ARRAY [0..4] OF WORD | [16#39,3(16#9),16#F] | Segments to display in upper screen |
| 4 | VAR | iScreenSelection | INT | 0 | Select the screen |



| Array element | Control data | Display |
|---|---|---|
| 1st | 16#39 |  |
| 2nd to 4th | 16#9 |  |
| 5th | 16#F |  |

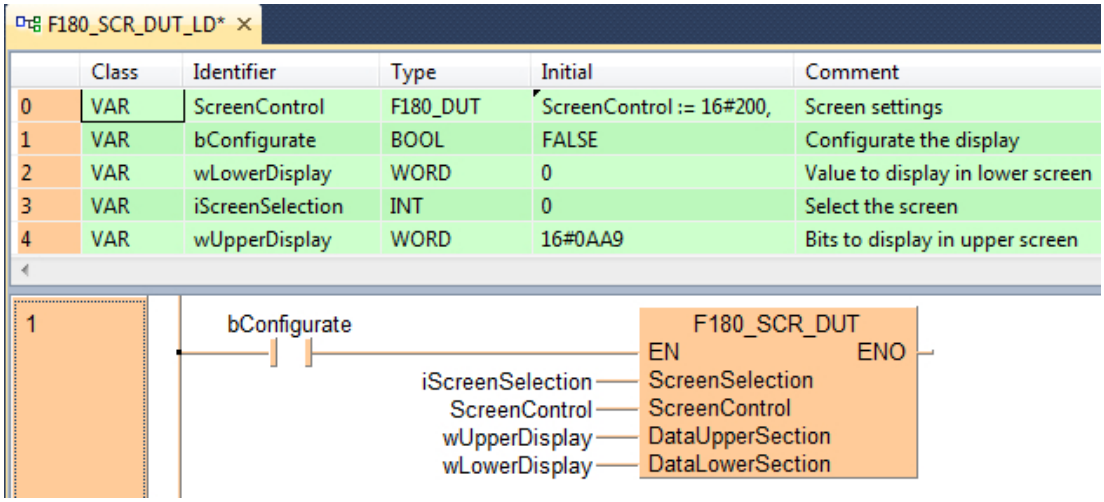
The segment data is displayed as follows:



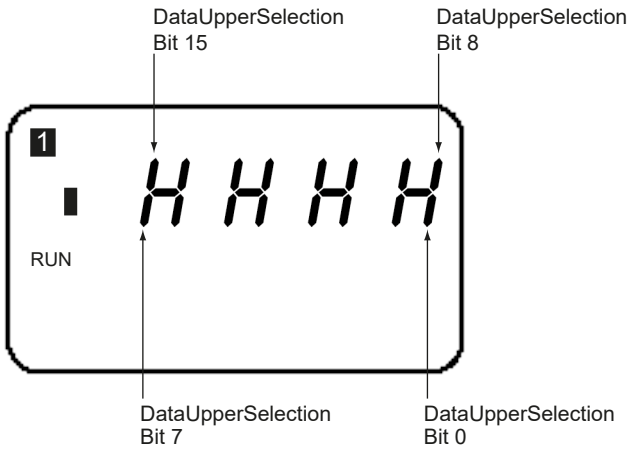

7-segment conversion table

## Display of bit data with F180

The bit data to be displayed is stored in the variable **wUpperDisplay**.

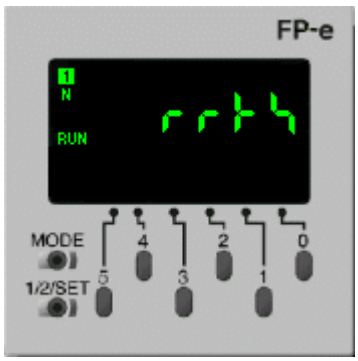



The word **wUpperDisplay** is connected to the input **DataUpperSection**.
DataUpperSelection is encoded as follows:




**DataLowerSection** is encoded in the same way.

The bit data above is displayed as follows:




## FP-e: short description of the front switches

Pressing the operation switch “0” to “4” for N mode 1st screen switches the mode to the change mode for SV0.

When the display selection switch "1/2/SET" is pressed for about one second, the data for the SV0 is changed and the data stops blinking.

- Data blinks in the change mode.
- Data which is out of the specified range (16-bit) cannot be written.

When the display selection switch "1/2/SET" is pressed, the current screen changes to the 2nd screen.

Pressing the "MODE" switch for about 2 seconds locks both the display selection switch and the operation switch. In this "LOCK" status, the display and data cannot be changed. In addition, the "LOCK" status is not cancelled even when the power turns ON/OFF.

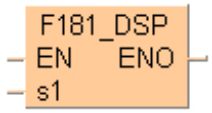
Pressing the "MODE" switch for about 2 seconds again unlocks the "LOCK" status. At this time, the "LOCK" display turns off.

For detailed information please refer to the technical manual of the FP-e (FP-e User's Manual, ARCT1F369). For PDF files, please refer to Panasonic Download Center

# F181_DSP

Screen change instruction

The FP-e display mode is changed to the one specified using **s1**.




## Parameters

Input

**s1**

Display mode and No. (0–7 can be specified).

| | |
|---|---|
| 0 | N mode 1st screen |
| 1 | N mode 2nd screen |
| 2 | S mode 1st screen |
| 3 | S mode 2nd screen |
| 4 | R mode 1st screen |
| 5 | R mode 2nd screen |
| 6 | I mode 1st screen |
| 7 | I mode 2nd screen |

(N=normal mode, S=switch mode, R=register mode, I=I/O monitor mode).

## Remarks

- If a value other than 0–7 is specified for **s1**, an operation error will occur.
- This instruction cannot be used during an interrupt program.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the value at **s1** is not in the range 0–7

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the value at **s1** is not in the range 0–7

## Example

### POU header

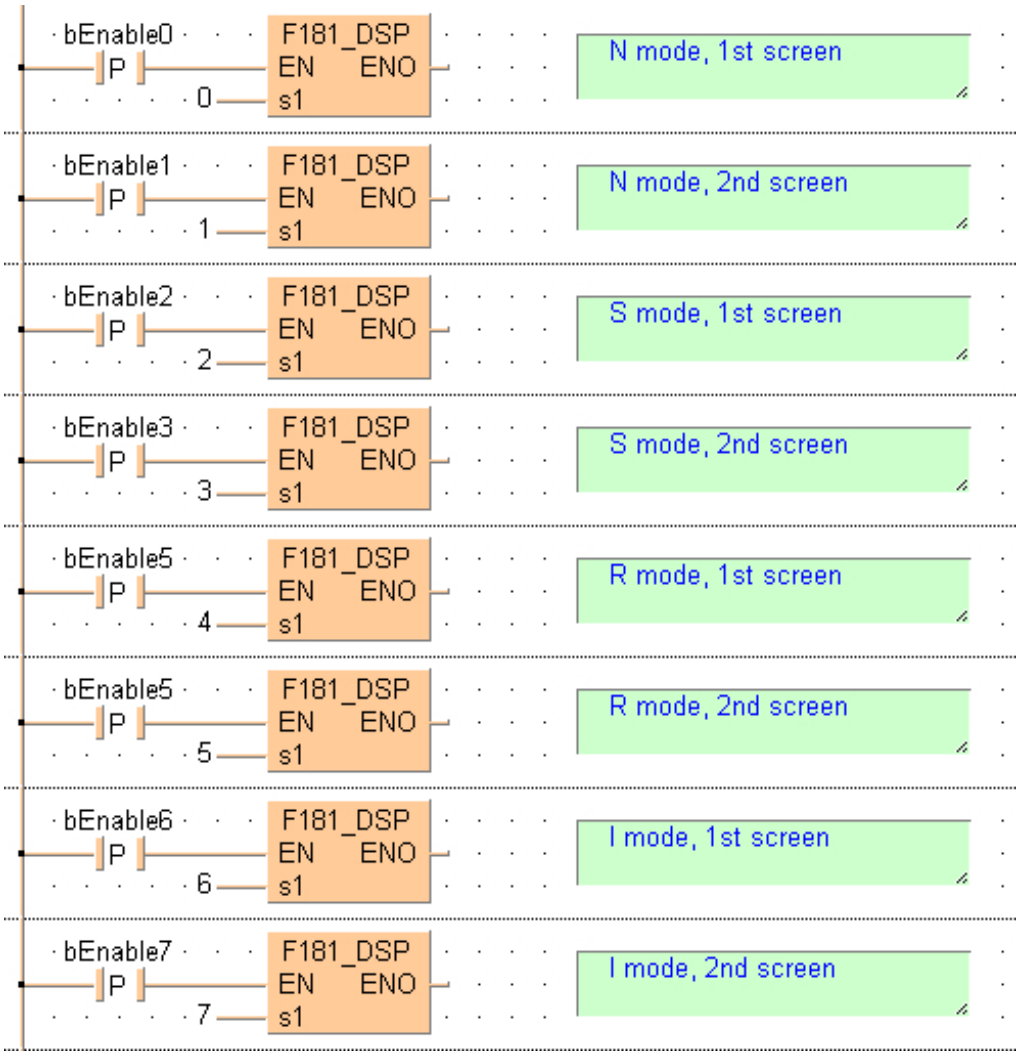
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnable0 | BOOL | FALSE |
| 1 | VAR | bEnable1 | BOOL | FALSE |
| 2 | VAR | bEnable2 | BOOL | FALSE |
| 3 | VAR | bEnable3 | BOOL | FALSE |
| 4 | VAR | bEnable4 | BOOL | FALSE |
| 5 | VAR | bEnable5 | BOOL | FALSE |
| 6 | VAR | bEnable6 | BOOL | FALSE |
| 7 | VAR | bEnable7 | BOOL | FALSE |

### POU body

According to the variable **Enable0** to **Enable7** that is set to TRUE, the function is executed and the FP-e is switched to the corresponding mode and the corresponding screen. (N=normal mode, S=switch mode, R=register mode, I=I/O monitor mode).

## LD body

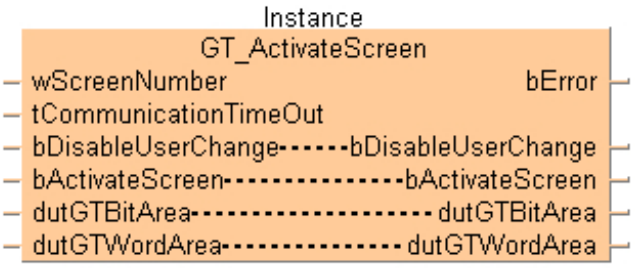

bEnable0
P
0
F181_DSP
EN
ENO
s1
N mode, 1st screen
bEnable1
P
1
F181_DSP
EN
ENO
s1
N mode, 2nd screen
bEnable2
P
2
F181_DSP
EN
ENO
s1
S mode, 1st screen
bEnable3
P
3
F181_DSP
EN
ENO
s1
S mode, 2nd screen
bEnable5
P
4
F181_DSP
EN
ENO
s1
R mode, 1st screen
bEnable5
P
5
F181_DSP
EN
ENO
s1
R mode, 2nd screen
bEnable6
P
6
F181_DSP
EN
ENO
s1
I mode, 1st screen
bEnable7
P
7
F181_DSP
EN
ENO
s1
I mode, 2nd screen

# 19 GT panel instructions

# GT_ActivateScreen

Control the GT panel screen

Function block to activate or change a specified GT screen from the PLC using the variables described in the table for data types.




## Parameters

Input

**wScreenNumber** (WORD)

Screen number

**tCommunicationTimeOut** (TIME)

Communication timeout

Input/output

**bDisableUserChange** (BOOL)

Disable screen change by touch operation on GT

**bActivateScreen** (BOOL)

Activate new screen

**dutGTBitArea** (**GT_CommunicationBitArea_DUT**)

GT basic communication bit area

**dutGTWordArea** (**GT_CommunicationWordArea_DUT**)

GT basic communication word area

Output

**bError** (BOOL)

Turns on when the screen is not switched within the communication timeout

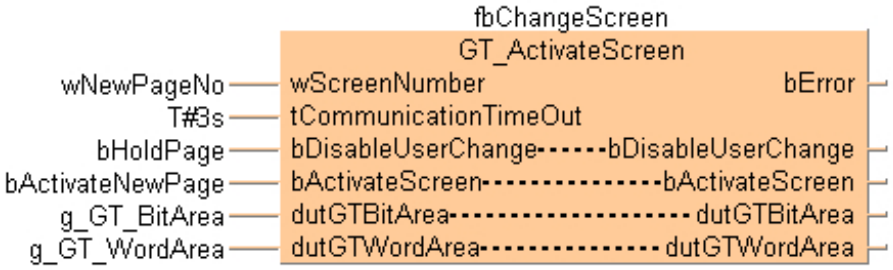
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_GT_WordArea | GT_BasicComWordArea | |
| 1 | VAR_EXTERNAL | g_GT_BitArea | GT_BasicComBitArea | |
| 2 | VAR | bActivateNewPage | BOOL | FALSE |
| 3 | VAR | wNewPageNo | WORD | 0 |
| 4 | VAR | g_bStartPage | BOOL | FALSE |
| 5 | VAR | fbChangeScreen | GT_CtrlActivateScreen | |
| 6 | VAR | bHoldPage | BOOL | FALSE |

### LD body



# GT_ChangeBacklightBrightness

Changes the backlight brightness of a GT panel

This instruction changes the backlight brightness of the GT Panel using the variables described in the table for data types.


## Parameters

Input

**iBrightness** (INT)

Brightness value 0–15

Input/output

**dutGTBitArea** (**GT_CommunicationBitArea_DUT**)

GT basic communication bit area

Output

**bError** (BOOL)

Turns on if the brightness value is out of range

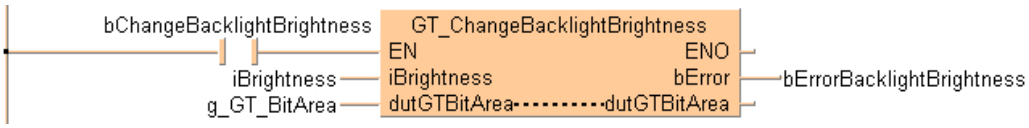
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_GT_BitArea | GT_BasicComBitArea | |
| 1 | VAR | iBrightness | INT | 0 |
| 2 | VAR | bErrorBacklightBrightness | BOOL | FALSE |
| 3 | VAR | bChangeBacklightBrightness | BOOL | FALSE |

## LD body


bChangeBacklightBrightness
GT_ChangeBacklightBrightness
EN
ENO
iBrightness
iBrightness
bError
bErrorBacklightBrightness
g_GT_BitArea
dutGTBitArea
dutGTBitArea

# 20 High-speed counter instructions

## 20.1 Introduction

Note

The high-speed counter and pulse output instructions can be used with the following FP Series PLCs: FP0, FP-e, FPΣ, FP-X, FP0R.

### Introduction

Control FPWIN Pro offers two concepts for programming with high-speed counter instructions:

- F instructions
- Tool instructions

The tool instructions are universal instructions which are supported by all PLC types of the FP series. They offer new and comfortable features including information functions for evaluating status flags and settings, control functions for configuring high-speed counters and pulse outputs, PLC-independent functions and DUTs, as well as variable channel numbers.
When should you use tool instructions instead of F instructions?

- You want to develop universal function blocks for libraries.
- You must program for different PLC types of the FP series.
- You're tired of setting control code bits and looking up available channel numbers.

However, the F instructions may be easier to use for beginners or users familiar with FPWIN GR.

Most of the information, which is accessible via information and control functions, is stored in special internal flags and special data registers. These flags and registers can also be accessed using PLC-independent system variables.
To take advantage of the features you prefer, the instructions of both libraries can be mixed.

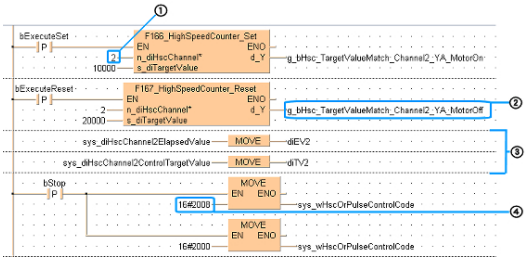
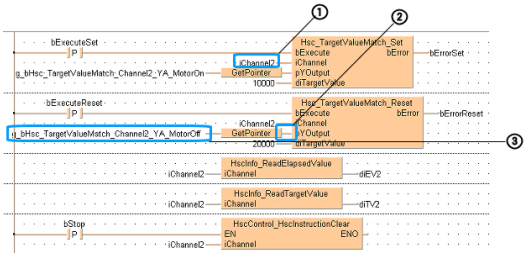
Note

When programming with the tool instructions, be sure to refer to the detailed information provided via the links to the related F instructions.

| Main features | F instructions | Tool instructions |
|---|---|---|
| Pre version 6.4 support | ● | |
| Use of inline functions | ● | |
| Use of FPWIN GR function names | ● | |
| Less code with constant channel numbers | ● | |
| Control codes | ● | |

| Main features | F instructions | Tool instructions |
|---|---|---|
| Control functions | | ● |
| Information functions | | ● |
| Variable channel numbers | | ● |
| Universal functions for all PLCs | | ● |
| For use in universal user function blocks | | ● |
| Common channel configuration DUT for all PLCs and all pulse output instructions | | ● |

## Comparison between programming with F instructions and Tool instructions

| F instructions | | Tool instructions | |
|---|---|---|---|
|  | |  | |
| (1) | Supports only constant channel numbers, in this example channel 2. | (1) | Supports variable channel numbers, in this example channel 2. |
| (2) | Outputs with explicit user addresses in the Y area | (2) | Access to outputs with explicit user addresses via a pointer variable. This pointer variable can also be applied via inputs of user-defined function blocks. |
| (3) | System variables are used to read special data registers for channel 2. | (3) | The name of the output variable **g_bHsc_TargetValueMatch_Channel2_YA_MotorOff** must follow a certain pattern, please refer to Hsc_TargetValueMatch_Set. |
| (4) | PLC-specific control code settings are required, e.g. for clearing a high-speed counter instruction | | |
| Conclusion:<br>• Depends on PLC type<br>• Constant channel number<br>• High maintenance effort, e.g. to change the channel number | | Conclusion:<br>• Independent of PLC type<br>• Variable channel number<br>• Self-explanatory function block names and variables<br>• Can be called by user-defined function blocks with variable channel numbers<br>• Channel number easy to change<br>• Requires more programming steps | |

## Typical applications

Use the high-speed counter instructions to count input pulses from sensors or encoders, and to turn outputs to TRUE or FALSE once a specified target value has been reached.

When used with a motor driver, the pulse output instructions enable typical positioning operations such as trapezoidal control, home return, and JOG operation.

## Specifications

The number of channels for the built-in high-speed counter and for pulse output, the counting range, the input and output numbers, as well as performance specifications differ depending on the PLC type. For details, please refer to the corresponding hardware manual.

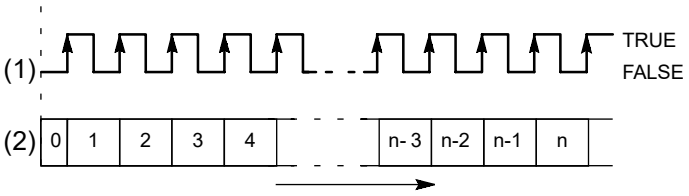
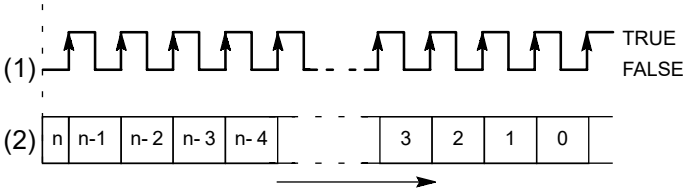
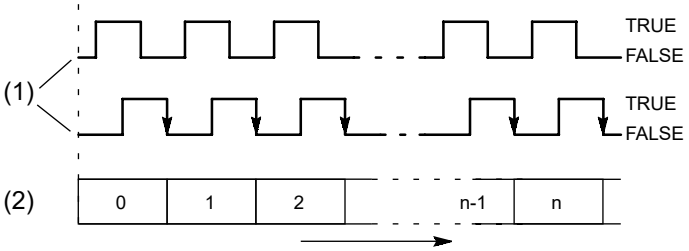
## Required system register settings

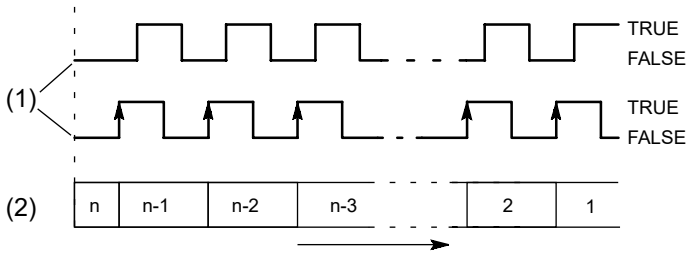
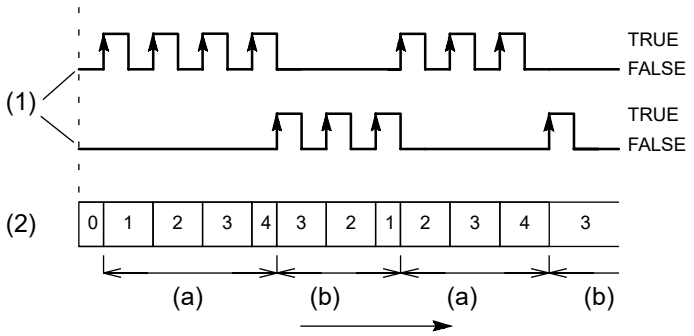
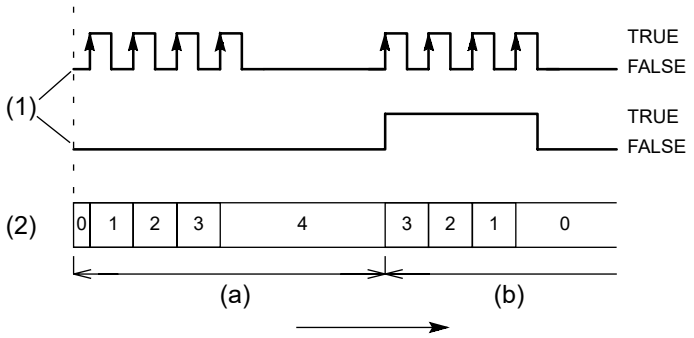
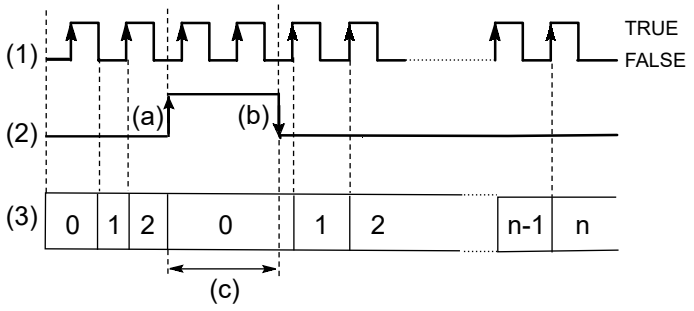
When using a high-speed counter instruction: Select the high-speed counter input for the desired channel in the system registers.

## Counter input modes

To specify the counter input mode, select the high-speed counter inputs in the system registers.

(1) High-speed counter input
(2) Counter value
(3) Reset input

| Input mode | Input signals |
|---|---|
| Incremental |  |
| Decremental |  |
| Two-phase | Incremental counting<br>  |

| Input mode | Input signals |
|---|---|
| | Decremental counting<br>  <br> |
| Incremental/decremental |  <br> |
| | (a) Increasing |
| | (b) Decreasing |
| Incremental/decremental control |  <br> |
| | (a) Increasing |
| | (b) Decreasing |
| Count for reset (incremental) |  <br> |
| | (a) Rising edge: count disabled, elapsed value cleared |
| | (b) Falling edge: count enabled |

| Input mode | Input signals | |
|---|---|---|
| | (c) | Count prohibited |
| | The reset at (3) is executed by the interruption at (a) (rising edge) and (b) (falling edge).The reset input can be enabled/disabled using bit 2 of sys_wHscOrPulseControlCode. | |

### Writing control codes

Control codes are used to perform special counter operations.

- When programming with F instructions:

  Use a MOVE instruction to write or read the control code to or from the special data register reserved for this code (DT90052 or DT9052, depending on the PLC type).The special data register where the high-speed counter and pulse output control code are stored can be accessed with the system variable **sys_wHscOrPulseControlCode**.

- When programming with tool instructions:

  Use universal high-speed counter control instructions and pulse output control instructions which apply to all PLC types to make control code settings. Use the high-speed counter information instructions and pulse output information instructions to monitor control code settings.

### Writing and reading the elapsed value

The elapsed value is stored as a double word in the special data registers.

- When programming with F instructions:
  - Access the special data registers using the system variable **sys_diHscChannelxElapsedValue** (where x=channel number).
  - The channel number is an input parameter of the high-speed counter or pulse output instruction. Most of the other parameters, e.g. speed and target value, can be specified using predefined DUTs. These DUTs can be found in the `FP Library`.
- When programming with tool instructions:
  - Use universal high-speed counter information and control instructions and pulse information and control instructions which apply to all PLC types to read and write the elapsed value.
  - The channel number and control code settings, e.g. CW/CCW, absolute or relative value control, or duty ratio are specified in a channel configuration DUT common to all PLC types. Other parameters, e.g. speed and target value can be applied directly to the instruction.

### Control flags

The high-speed counter and pulse output status is stored in special internal flags. To access the special internal flags, use the PLC-independent system variables.

When a high-speed counter instruction is executed, the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) for the channel used turns to TRUE. No other high-

speed counter instruction using the same channel can be executed as long as the control flag is TRUE.

When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.

- FP-X, FP0R:

  The high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) are assigned to different special internal flags.

- FP-Sigma, FP0, FP-e:

  The high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE. The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

## 20.2 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

Advantages of IEC instructions

Advantages of FP instructions

### 20.2.1 Writing the high-speed counter control code

The special data register where the high-speed counter and pulse output control code are stored can be accessed with the system variable **sys_wHscOrPulseControlCode**. (The system variable sys_wHscOrPulseControlCode corresponds to special data register DT90052.)

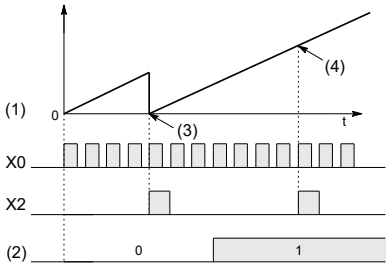
Operations performed by the high-speed counter control code

The control code settings for each channel can be monitored using the system variables **sys_wHscChannelxControlCode** or **sys_wPulseChannelxControlCode** (where x=channel number).The settings of this system variable remain unchanged until another setting operation is executed.

#### Cancelling high-speed counter instructions (bit 3)

To cancel execution of an instruction, set bit 3 of the data register storing the high-speed counter control code (**sys_wHscOrPulseControlCode**) to TRUE. The high-speed counter control flag then changes to FALSE. To re-enable execution of the high-speed counter instruction, reset bit 3 to FALSE.

## Enabling/disabling the reset input (hardware reset) of the high-speed counter (bit 2)



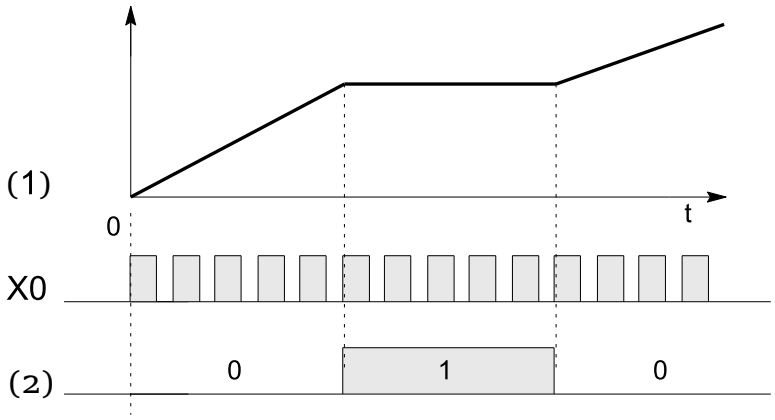

X0 High-speed counter input

(1) Elapsed value
(2) Bit 2 of high-speed counter control code (enable/disable reset input)
(3) Elapsed value is reset to 0
(4) Reset not possible

When bit 2 of the control code is set to TRUE, a hardware reset using the reset input specified in the system registers is not possible. Counting will continue even if the reset input has turned to TRUE. The hardware reset is disabled until bit 2 is reset to 0.

## Enabling/disabling counting operations (bit 1)

Count input control operation

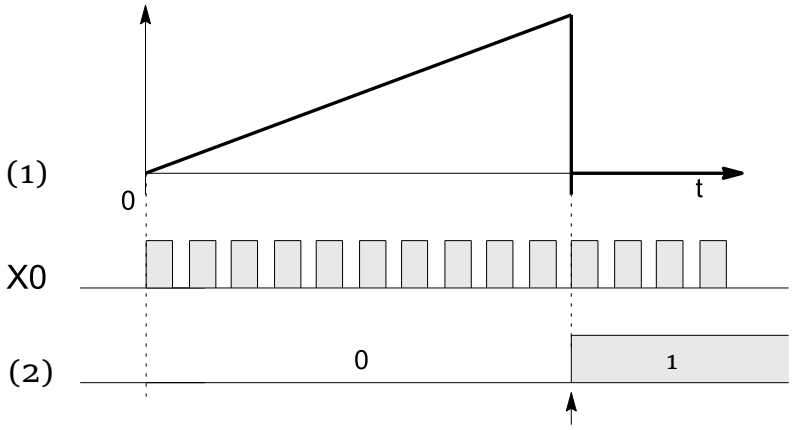



X0 High-speed counter input

(1) Elapsed value
(2) Bit 1 of high-speed counter control code (count)

When bit 1 of the control code is set to TRUE, counting is prohibited and the elapsed value keeps its current value. Counting is continued when bit 1 is reset to FALSE.

## Resetting the elapsed value (software reset) of the high-speed counter to 0 (bit 0)




X0 High-speed counter input

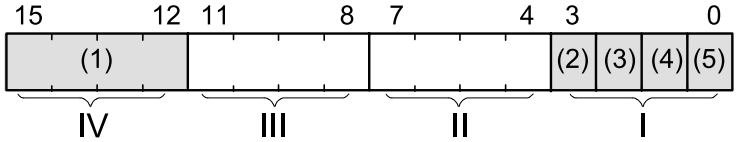
(1) Elapsed value

(2) Bit 0 of high-speed counter control code (software reset)

When bit 0 of the control code is set to TRUE, a software reset is performed and the elapsed value is set to 0. The elapsed value keeps the value 0 until bit 0 is reset to FALSE.

## Description for FP-Sigma, FP-X, FP0R

Bits 0–15 of the control code are allocated in groups of four. The bit setting in each group is represented by a hex number (e.g. 00020000 0000 1001 = 16#2009).




(1) Channel number (channel n: 16#n)

(2) Cancel high-speed counter instruction (bit 3)
0: continue/1: cancel

(3) Reset input (bit 2) (see note)
0: enabled/1: disabled

(4) Count (bit 1)
0: permit/1: prohibit

(5) Reset elapsed value to 0 (bit 0)
0: no/1: yes

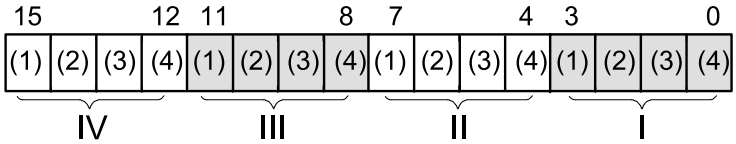
Example: 16#2009

| Group | Value | Description |
|---|---|---|
| IV | 2 | Channel number: 2 |
| III | 0 | (fixed) |
| II | 0 | (fixed) |
| I | 9 | Hex 9 corresponds to binary 1001 |

| Group | Value | Description | |
|---|---|---|---|
| | | Cancel high-speed counter instruction: cancel (bit 3) | 1 |
| | | Reset input: enabled (bit 2) | 0 |
| | | Count: permit (bit 1) | 0 |
| | | Reset elapsed value to 0: yes (bit 0) | 1 |

## Description for FP0, FP-e

Bits 0–15 of the control code are allocated in groups of four, each group containing the settings for one channel. The bit setting in each group is represented by a hex number (e.g. 0000 0000 1001 0000 = 16#90).




(1) Cancel high-speed counter instruction (bit 3)
0: continue/1: cancel

(2) Reset input (bit 2) (see note)
0: enabled/1: disabled

(3) Count (bit 1)
0: permit/1: prohibit

(4) Reset elapsed value to 0 (bit 0)
0: no/1: yes

| Group | IV | III | II | I |
|---|---|---|---|---|
| Channel | 3 | 2 | 1 | 0 |

### Example: 16#90

| Group | Value | Description | |
|---|---|---|---|
| IV | 0 | – | |
| III | 0 | – | |
| II | 9 | Channel number: 1<br>Hex 9 corresponds to binary 1001 | |
| | | Cancel high-speed counter instruction: cancel (bit 3) | 1 |
| | | Reset input: enabled (bit 2) | 0 |
| | | Count: permit (bit 1) | 0 |
| | | Reset elapsed value to 0: yes (bit 0) | 1 |
| I | 0 | – | |

Note

Turning the reset input to TRUE, sets the elapsed value to 0. Use the reset input setting (bit 2) to disable the reset input allocated in the system registers.

## Example: software reset for channel 0

### POU header

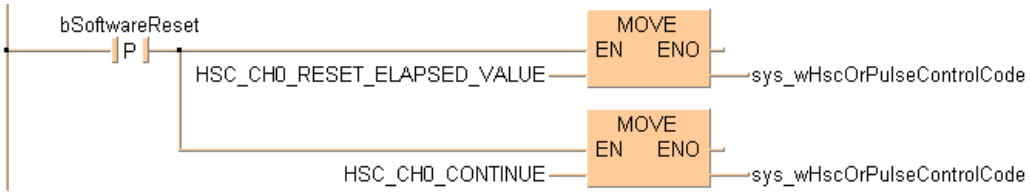
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bSoftwareReset | BOOL | FALSE | Activates the function |
| 1 | VAR_CONSTANT | HSC_CH0_RESET_ELAPSED_VALUE | WORD | 16#0001 | Resets elapsed value of channel 0 |
| 2 | VAR_CONSTANT | HSC_CH0_CONTINUE | WORD | 16#0000 | Continues counting in channel 0 |

### POU body

The reset is performed in step 1, and 0 is entered just after that in step 2 to start counting. A reset alone does not start counting.

### LD body




## Example: software reset for channel 1 (FP-SIGMA, FP-X, FP0R)

### POU header

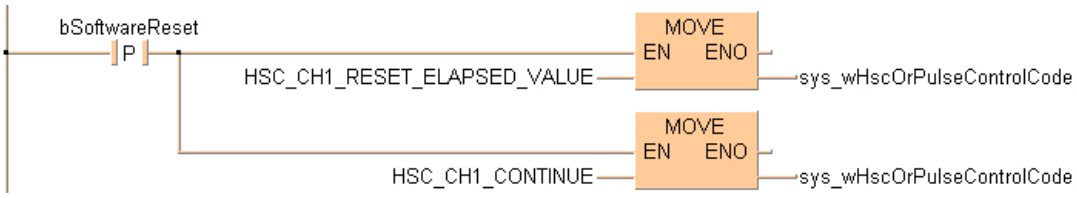
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bSoftwareReset | BOOL | FALSE | Activates the function |
| 1 | VAR_CONSTANT | HSC_CH1_RESET_ELAPSED_VALUE | WORD | 16#1001 | Resets elapsed value of channel 1 |
| 2 | VAR_CONSTANT | HSC_CH1_CONTINUE | WORD | 16#1000 | Continues counting in channel 1 |

### POU body

The reset is performed in step 1, and 0 is entered just after that in step 2 to start counting. A reset alone does not start counting.

### LD body




## 20.2.2 High-speed counter: writing and reading the elapsed value

The elapsed value is stored as a double word in the special data registers.

- When programming with F instructions:Access the special data registers using the system variable **sys_diHscChannelxElapsedValue** (where x=channel number).
- When programming with tool instructions:Use universal high-speed counter information and control instructions and pulse information and control instructions which apply to all PLC types to read and write the elapsed value.

System variables for memory areas used:

| PLC type | High-speed counter: elapsed value of channel | System variables |
|---|---|---|
| FP-Sigma, FP-e | 0-3 | **sys_diHscChannel0ElapsedValue-sys_diHscChannel3ElapsedValue** |
| FP-X, Transistor types | 0-7 | **sys_diHscChannel0ElapsedValue-sys_diHscChannel7ElapsedValue** |
| FP-X, Relay types | 0-B | **sys_diHscChannel0ElapsedValue-sys_diHscChannelBElapsedValue** |
| FP0R | 0-5 | **sys_diHscChannel0ElapsedValue-sys_diHscChannel5ElapsedValue** |

### Example writing the elapsed value into the high-speed counter

The first example shows how to write an initial value (elapsed value) into the high-speed counter. The second example shows how to read an elapsed value and copy it to a variable.

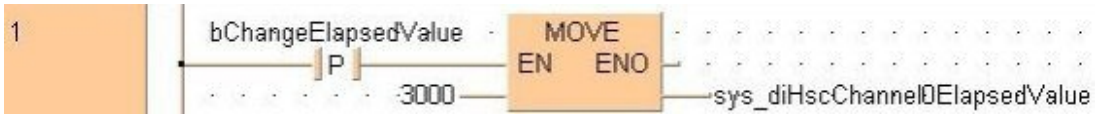
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bChangeElapsedValue | BOOL | FALSE | Changes the elapsed value |

### LD body

An initial value of 3000 (elapsed value) is written into channel 0 of the high-speed counter.




### Example2

Reading the elapsed value and copying it to a variable

### POU header

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bReadElapsedValue | BOOL | FALSE | Reads the elapsed value |
| 1 | VAR | diElapsedValue | DINT | 0 | Outputs elapsed value |

### POU body

The elapsed value of the high-speed counter is read from channel 0 of the high-speed counter and copied to the variable **diElapsedValue**.

### LD body


## 20.2.2.2 High-speed counter: writing and reading the elapsed value

The elapsed value is stored as a double word in the special data registers.

- When programming with F instructions:Access the special data registers using the system variable **sys_diHscChannelxElapsedValue** (where x=channel number).
- When programming with tool instructions:Use universal high-speed counter information and control instructions and pulse information and control instructions which apply to all PLC types to read and write the elapsed value.

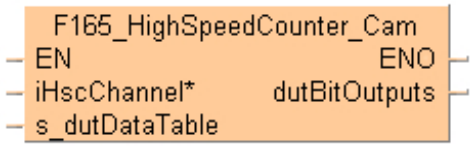
System variables for memory areas used:

| PLC type | High-speed counter: elapsed value of channel | System variables |
|---|---|---|
| FP-Sigma, FP-e | 0-3 | **sys_diHscChannel0ElapsedValue-sys_diHscChannel3ElapsedValue** |
| FP-X, Transistor types | 0-7 | **sys_diHscChannel0ElapsedValue-sys_diHscChannel7ElapsedValue** |
| FP-X, Relay types | 0-B | **sys_diHscChannel0ElapsedValue-sys_diHscChannelBElapsedValue** |
| FP0R | 0-5 | **sys_diHscChannel0ElapsedValue-sys_diHscChannel5ElapsedValue** |

# F165_HighSpeedCounter_Cam

Cam control

This instruction performs cam control according to the parameters in the specified DUT.




Refer to the parameter description according to the PLC type connected:

FP0R: **F165_HighSpeedCounter_Cam**
FP0H: **F165_HighSpeedCounter_Cam**

DUT sample used for the data table

FP0R: F165_HighSpeedCounter_Cam_8_Values_DUT
FP0H: F165_HighSpeedCounter_Cam_8_Values_OnOff_DUT

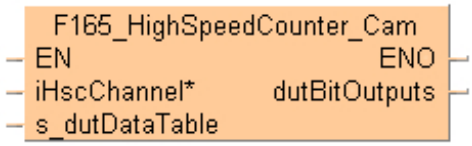
Differences of **F165_HighSpeedCounter_Cam** between FP0R and FP0H.

| Feature | FP0R | FP0H |
|---|---|---|
| High-speed counter channels | 0–5 | 0–3 |
| Target values | ON values<br>Maximum of 31 target values | ON values and OFF values<br>Maximum of 32 target values |
| Word address for outputs | WR | WR, WL, WY |

# F165_HighSpeedCounter_Cam

Cam control for FP0R

This instruction performs cam control according to the parameters in the specified DUT with a maximum of 31 target values for the high-speed counter. An interrupt program can be executed whenever the elapsed value matches one of the target values.




## Parameters

Input

**iHscChannel*** (INT)

High-speed counter channel: 0–5

**s_dutDataTable** (ANY_DUT)

Starting address of area containing the data table
Sample: **F165_HighSpeedCounter_Cam_8_Values_DUT**

Output

**dutBitOutputs** (ANY_DUT)

Starting address (WR) of area containing the output word address, e.g. **BOOL32_OVERLAPPING_DUT**. Select the size (16 or 32 bits) according to the number set with **diNumberOfTargetValuesAndOutputRelays**.

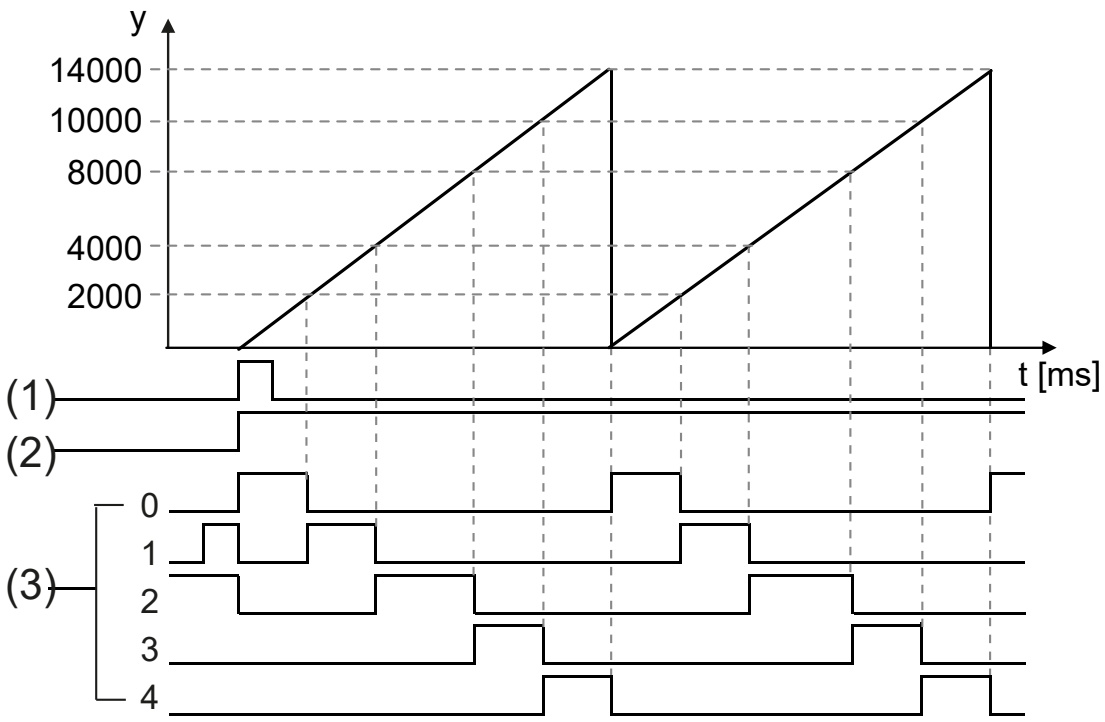
## Remarks

Create your own DUT using the following DUT as a sample:
**F165_HighSpeedCounter_Cam_8_Values_DUT**

The following parameters can be specified in the DUT:

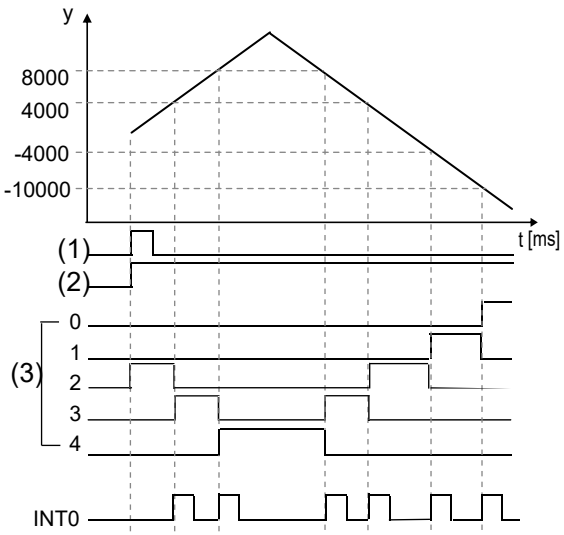
- Control code
- Word address for outputs
- Number of target values
- Target value1
- ...
- Target value**n**
- Maximum target value

### Characteristics of cam control




| | | | |
|---|---|---|---|
| y | Elapsed value of high-speed counter | 14000 | Maximum target value |
| (1) | Execution condition | 10000 | Target value4 |
| (2) | High-speed counter control flag | 8000 | Target value3 |
| (3) | Output0-4 | 4000 | Target value2 |
| | | 2000 | Target value1 |

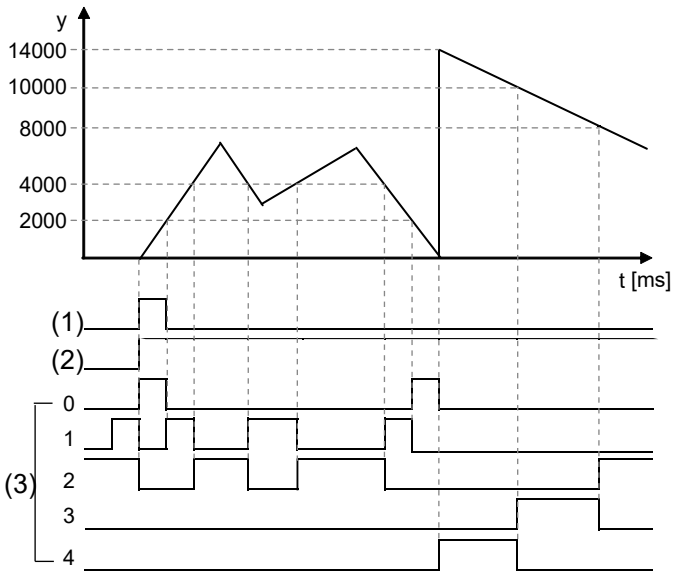
- Whenever the elapsed value is in the target value area n to n+1 (incremental counting) or n+1 to n, (decremental counting), the corresponding output n is TRUE.
- In the example above, maximum target value control has been enabled. When the elapsed value reaches the maximum target value, the elapsed value is reset to 0 and counting restarts.
- Specify the word address of the outputs in an overlapping DUT, e.g. **BOOL32_OVERLAPPING_DUT**, and apply this DUT at **dutBitOutputs**.
- A maximum of 31 target values can be specified.
- The target values must be arranged in ascending order. No value may be used twice.
- When the instruction starts, all outputs are FALSE, except for output 0, which turns to TRUE, provided that the elapsed value is smaller than target value 1. Otherwise, the output corresponding to the target value area turns to TRUE. Example: If the current value is between target value 2 = -4000 and target value 3 = +4000, output 2 is TRUE. In the following example maximum target value control has been disabled. When the elapsed value reaches the last target value, counting continues and the elapsed value is not reset to 0.




| | | | |
|---|---|---|---|
| y | Elapsed value of high-speed counter | 8000 | Target value4 |
| (1) | Execution condition | 4000 | Target value3 |
| (2) | High-speed counter control flag | -4000 | Target value2 |
| (3) | Output0-4 | -10000 | Target value1 |
| INT0 | Interrupt program 0 | | |

### Maximum target value control

- The instruction can be executed using maximum target value control to reset the elapsed value to 0 when the maximum target value has been reached. Maximum target value control can be enabled in the control code of **F165_HighSpeedCounter_Cam_8_Values_DUT**. Instead of using maximum target value control, the elapsed value can also be reset using a reset input or a software reset.
- To perform maximum target value control, positive integer numbers must be specified for all target values.
- Incremental and decremental counting with maximum target value control:



| | | | |
|---|---|---|---|
| y | Elapsed value of high-speed counter | 14000 | Maximum target value |
| (1) | Execution condition | 10000 | Target value4 |
| (2) | High-speed counter control flag | 8000 | Target value3 |
| (3) | Output0-4 | 4000 | Target value2 |
| | | 2000 | Target value1 |

## Overview:

| **Maximum target value control:** | **enabled** | **disabled (see note)** |
|---|---|---|
| Incremental counting:<br>The pointer of the data table moves from target value 1 to the last target value. | When the elapsed value reaches the maximum target value:<br>• the pointer returns to target value 1<br>• output 0 turns to TRUE<br>• the elapsed value is set to 0 | When the elapsed value reaches the last target value:<br>• the pointer returns to target value 1<br>• output 0 turns to TRUE<br>• the elapsed value continues to increment and restarts at the minimum value of the ring counter |
| Decremental counting:<br>The pointer of the data table moves from the last target value to target value 1. | When the elapsed value reaches the value -1:<br>• the pointer returns to the last target value<br>• the output corresponding to the last target value turns to TRUE<br>• the elapsed value is set to the maximum target value | When the elapsed value reaches the value -1:<br>• the pointer returns to target value **n**<br>• the output corresponding to the last target value turns to TRUE<br>• the elapsed value continues to decrement and restarts at the maximum value of the ring counter |

Note

Provided that neither a reset input nor a software reset is being used.

## Hardware reset operation

| **Channel** | **Hardware reset input** |
|---|---|
| 0 | X2 |
| 1 | |
| 2 | X5 |
| 3 | |

### Interrupt operation

The interrupt program will be executed when the elapsed value matches the target value. Any interrupt that has been entered into the Tasks list is automatically enabled.A special interrupt program number is assigned to each channel number.

| Channel | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Interrupt program | 0 | 1 | 3 | 4 | 6 | 7 |

### General programming information

- Select the high-speed counter input for the desired channel in the system registers.
- When a high-speed counter instruction is executed, the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) for the channel used turns to TRUE. No other high-speed counter instruction using the same channel can be executed as long as the control flag is TRUE.
- To cancel execution of an instruction, set bit 3 of the data register storing the high-speed counter control code (**sys_wHscOrPulseControlCode**) to TRUE. The high-speed counter control flag then changes to FALSE. To re-enable execution of the high-speed counter instruction, reset bit 3 to FALSE.
- Rewriting the elapsed value for the channel used during the execution of the instruction may cause an unexpected operation.
- Make sure the time span between adjacent target values is greater than 1ms.
- If the instruction is executed in the main program, make sure the minimum time span between adjacent target values is greater than the scan time.
- If the instruction is executed in an interrupt program, make sure the minimum time span between adjacent target values is greater than the maximum execution time of the interrupt program.
- This instruction can be executed simultaneously on a maximum of two channels.
- When using a reset input or a software reset, make sure target value 1 is an integer and ≥1.
- When maximum target value control is used together with a reset input or software reset, be careful not to use them at the same time.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if channel number or values of the data table are outside the permissible range
- if high-speed counter has not been set in the system registers
- if target value > maximum target value.
- if target value = 0.

- if target values are not arranged in ascending order

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if channel number or values of the data table are outside the permissible range
- if high-speed counter has not been set in the system registers
- if target value > maximum target value.
- if target value = 0.
- if target values are not arranged in ascending order

## Example with maximum target value control

### DUT

The DUT **F165_HighSpeedCounter_Cam_8_Values_DUT** is predefined in the FP Library and can be used as a sample.

| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | dwCamControlCode | DWORD | 16#0010 | 10: with maximum target value |
| 1 | diAddressOffsetInWR | DINT | 0 | |
| 2 | diNumberOfTargetValuesAndOutputRelays | DINT | 4 | |
| 3 | diTargetValue_1 | DINT | 2000 | |
| 4 | diTargetValue_2 | DINT | 4000 | |
| 5 | diTargetValue_3 | DINT | 8000 | |
| 6 | diTargetValue_4 | DINT | 10000 | |
| 7 | diMaximumTargetValue | DINT | 14000 | |

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

| | Class | Identifier | FP Address | IEC Address | Type |
|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | WR0_bits_F165_CAM_Examples | WR1 | %MW0.1 | BOOL16_OVERLAPPING_DUT |

### POU header

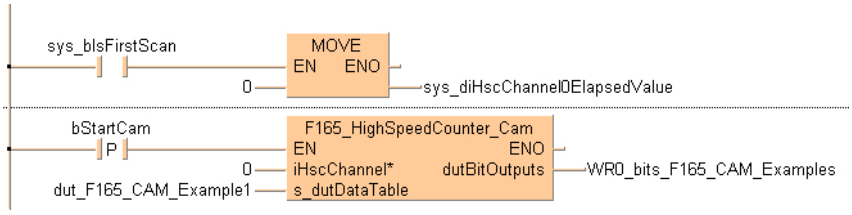
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | dut_F165_CAM_Example1 | F165_Cam_Example1_4_Values_DUT | |
| 1 | VAR | bStartCam | BOOL | FALSE |
| 2 | VAR_EXTERNAL | WR0_bits_F165_CAM_Examples | BOOL16_OVERLAPPING_DUT | |

### POU body

When the variable **bStartCam** turns to TRUE, the function is carried out.

### LD body




## Example without maximum target value control

### DUT

The DUT **F165_HighSpeedCounter_Cam_8_Values_DUT** is predefined in the FP `Library` and can be used as a sample.

| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | dwCamControlCode | DWORD | 16#0000 | 00: without maximum target value |
| 1 | diAddressOffsetInWR | DINT | 0 | |
| 2 | diNumberOfTargetValuesAndOutputRelays | DINT | 4 | |
| 3 | diTargetValue_1 | DINT | -10000 | |
| 4 | diTargetValue_2 | DINT | -4000 | |
| 5 | diTargetValue_3 | DINT | 4000 | |
| 6 | diTargetValue_4 | DINT | 8000 | |
| 7 | diMaximumTargetValue | DINT | 0 | |

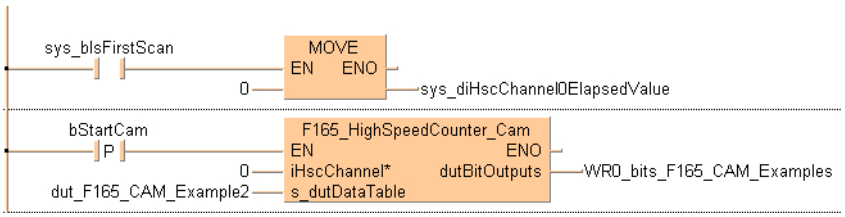
### POU header

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStartCam | BOOL | FALSE |
| 1 | VAR | dut_F165_CAM_Example2 | F165_Cam_Example2_4_Values_DUT | |
| 2 | VAR_EXTERNAL | WR0_bits_F165_CAM_Examples | BOOL16_OVERLAPPING_DUT | |

### POU body

When the variable **bStartCam** is set to TRUE, the function is carried out.

### LD body

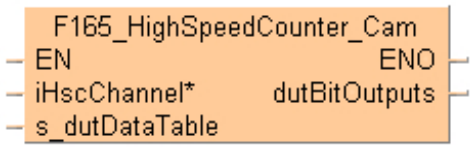


# F165_HighSpeedCounter_Cam

Cam control for FP0H

This instruction performs cam control as specified by the parameters in the specified DUT with a maximum of 32 target values according to the elapsed value of the high-speed counter. For every single cam output, the target value is set as a pair with an ON and an OFF set value.

An interrupt program can be executed whenever the elapsed value matches one of the target values.




## Parameters

Input

**iHscChannel*** (INT)

High-speed counter channel: 0–3

**s_dutDataTable** (ANY_DUT)

Starting address of area containing the data table

Output

**dutBitOutputs** (ANY_DUT)

Starting address (WR, WL or WY) of area containing the output word address, e.g. **BOOL32_OVERLAPPING_DUT**. Select the size (16 or 32 bits) according to the number set with **diNumberOfTargetValuesAndOutputRelays**.

## Remarks

- Input

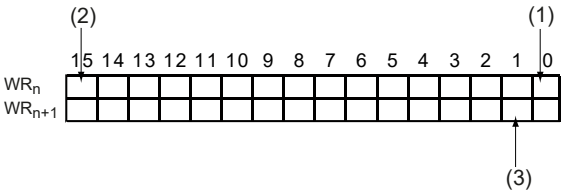
  Create your own DUT using the following DUT as a sample:
  F165_HighSpeedCounter_Cam_8_Values_OnOff_DUT

  The following parameters can be specified in the DUT:
  - Control code
  - Word address for outputs
  - Number of target values

    (ON/OFF set values from F165_HighSpeedCounter_Target_Values_OnOff_DUT)
  - Maximum target value

Setting range: 1–2147483646 (16#1–16#7FFFFFFE)

- Output
  - If the number of target values is in the range of 1–16, one word is used. If the number of target values is in the range of 17–32, two words are used.
  - Example: When the output address is set to "Internal flag", the starting word number of output address is set to "0", and the number of target values is set to "32", R0 to R1F are allocated as the address for the cam output.

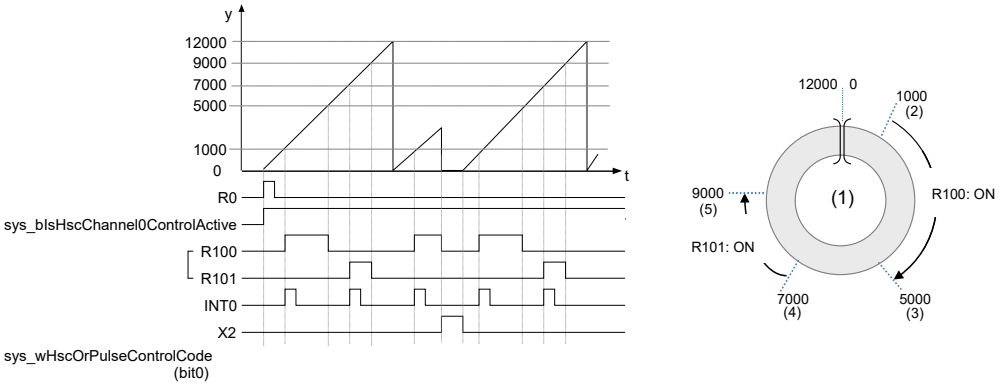



(1) When the elapsed value reaches the target value 1, R0 turns ON or OFF.
(2) When the elapsed value reaches the target value 16, RF turns ON or OFF.
(3) When the elapsed value reaches the target value 18, R11 turns ON or OFF.

Note

When you have specified the output flag (Y), values are output both to the CPU output and to the operation memories.

- Example




(1) Elapsed value
(2) Target value 1; ON set value: 1000
1000–4999 R100: ON
(3) Target value; OFF set value: 5000
(4) Target value 2; ON set value: 7000
7000–8999 R101: ON
(5) Target value; OFF set value: 9000

## Maximum target value control

- With **F165_HighSpeedCounter_Cam**, it is possible to perform the control with a specified maximum target. The settings for enabling/disabling the maximum target value control and the maximum target value are specified in the data table.
- The data table varies in the range of 12 to 138 words depending on the number of target values and the specified maximum target value setting.

Note

The maximum target value of the data table end is valid only when the target value control is set to 16#0010 (with maximum value) in **dwCamControlCode**. This setting can be omitted when the target value control is set to 16#0000 (without maximum value).

| | | **Enabled** | **Disabled** |
|---|---|---|---|
| Counting range | | 0 to target value | Negative minimum value to positive maximum value |
| Operation when the counting range is exceeded | Incremental counting: | When the elapsed value exceeds the target value, it is set to 0. | When the elapsed value exceeds the positive maximum value, it returns to the negative minimum value. |
| | Decremental counting:<br>The pointer of the data table moves from the last target value to target value 1. | When the elapsed value falls below 0, it is set to the target value. | When the elapsed value falls below the negative minimum value, it returns to the positive maximum value. |

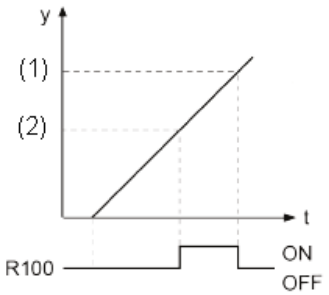
## Specification of target values

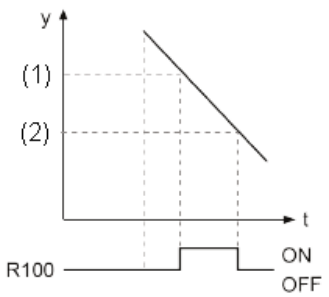
The output varies depending on the ON set value and OFF set value.

- OFF set value > ON set value

  When the elapsed value is larger than or equal to the ON set value and smaller than the OFF set value, the corresponding output bit turns on. When the elapsed value is out of range, the corresponding bit turns off.

| **Incremental counting:** | **Decremental counting:** |
|---|---|
| OFF set value (1) > ON set value (2) | |

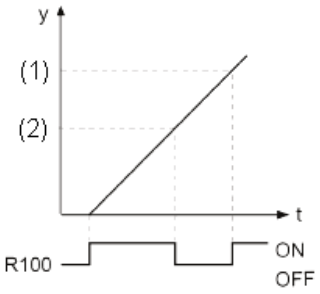


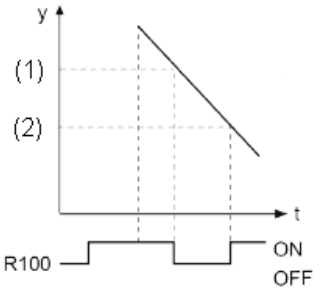




- ON set value > OFF set value

  When the elapsed value is smaller than the ON set value and larger than or equal to the OFF set value, the corresponding output bit turns off. When the elapsed value is out of range, the corresponding bit turns on.

| Incremental counting: | Decremental counting: |
|---|---|
| ON set value (1) > OFF set value (2) | |

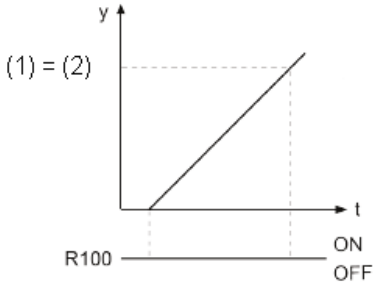


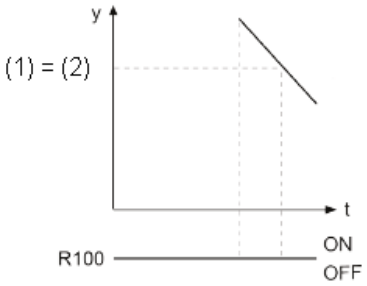




- ON set value = OFF set value

  When the elapsed value is out of range, the corresponding bit turns off.

| Incremental counting: | Decremental counting: |
|---|---|
| ON set value (1) = OFF set value (2) | |




## General programming information

- Select the high-speed counter input for the desired channel in the system registers.
- When a high-speed counter instruction is executed, the high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) for the channel used turns to TRUE. No other high-speed counter instruction using the same channel can be executed as long as the control flag is TRUE.
- This instruction can be executed simultaneously on a maximum of two channels.
- To cancel execution of an instruction, set bit 3 of the data register storing the high-speed counter control code (**sys_wHscOrPulseControlCode**) to TRUE. The high-speed counter control flag then changes to FALSE. To re-enable execution of the high-speed counter instruction, reset bit 3 to FALSE. When you set bit 3 of **sys_wHscOrPulseControlCode** to TRUE, this also disables the maximum target value control. When the maximum target value control is stopped, the cam output is held and the high-speed counter continues counting.
- Reset or preset the high-speed counter elapsed value before activating the instruction.
- Rewriting the elapsed value for the channel used during the execution of the instruction may cause an unexpected operation.

- If the instruction is executed in the main program, make sure the minimum time span between adjacent target values is greater than the scan time.
- If the instruction is executed in an interrupt program, make sure the minimum time span between adjacent target values is greater than the maximum execution time of the interrupt program.
- When using a reset input or a software reset, make sure target value 1 is an integer and ≥ 1.
- When maximum target value control is used together with a reset input or software reset, be careful not to use them at the same time.
- When the hardware reset or software reset is executed during the high-speed counter control, the high-speed counter elapsed value is reset to 0. The output allocated to the cam output will be the output according to the elapsed value 0.
- It is also possible to start the interrupt program INTn every time the elapsed value reaches each target value. For this operation, the activation of the interrupt program should be permitted by the interrupt control instruction **ICTL**.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if channel number or values of the data table are outside the permissible range
- if high-speed counter has not been set in the system registers
- if target value > maximum target value.
- if target value = 0.
- if target values are not arranged in ascending order

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if channel number or values of the data table are outside the permissible range
- if high-speed counter has not been set in the system registers
- if target value > maximum target value.
- if target value = 0.
- if target values are not arranged in ascending order

## Example using CH0 of FP0H

### DUT

The DUT **F165_HighSpeedCounter_Cam_8_Values_OnOff_DUT** is predefined in the FP Library and can be used as a sample.

## Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

| | Class | Identifier | FP address | IEC address | Type |
|---|---|---|---|---|---|
| 1 | VAR_GLOBAL | g_dutCamControl_WY0 | WY0 | %QW0 | BOOL32_OVERLAPPING_DUT |

## POU header

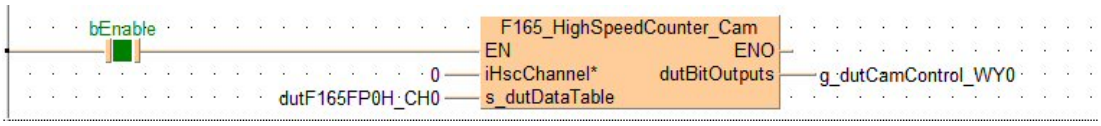
All input and output variables used for programming this function have been declared in the POU header.  The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | dutF165FP0H_CH0 | F165_HighSpeedCounter_Cam_8_Values_OnOff_DUT | |
| 3 | VAR_EXTERNAL | g_dutCamControl_WY0 | BOOL32_OVERLAPPING_DUT | |

## POU body

When the variable **bEnable** is set to TRUE, the function is executed.

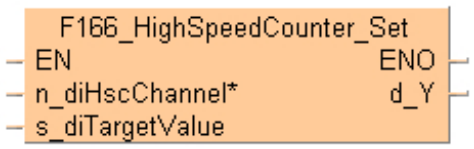
## LD body



# F166_HighSpeedCounter_Set

Target value match ON (high-speed counter)

If the elapsed value of the high-speed counter matches the target value, an interrupt process immediately turns the specified output to TRUE.




## Parameters

Input

**n_diHscChannel** (DINT)

High-speed counter channel:
FP-Σ: 0–3
FP-X R: 0–11
FP-X T: 0–7
FP0: 0–3
**F168_PulseOutput_Trapezoidal**: 0–3
**F171_PulseOutput_Trapezoidal**: 0–5

**s_diTargetValue** (DINT)

specify a 32-bit data value for the target value within the following range:
FP0, **F168_PulseOutput_Trapezoidal**: -838808–+8388607
FPΣ, FP-X, **F171_PulseOutput_Trapezoidal**: -2147483467–+2147483648

Output

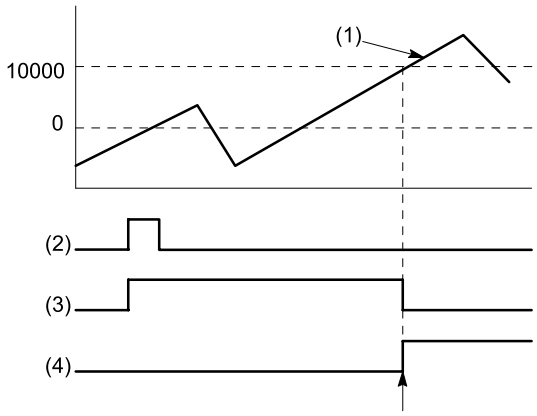
**d_Y** (BOOL)

output which turns to TRUE when the elapsed value matches the target value:
FP-Σ, FP0, **F168_PulseOutput_Trapezoidal**: Y0–Y7
FP-Σ (V3.1 or higher), **F171_PulseOutput_Trapezoidal**: Y0–Y1F
FP-X: Y0–Y29F

## Remarks

Characteristics of target value match ON control

Target value: 10000




(1) Elapsed value of high-speed counter
(2) Execution condition
(3) High-speed counter control flag
(4) PLC output

The PLC output turns to TRUE when the elapsed value matches the target value. In addition, the high-speed counter control flag turns to FALSE and the instruction is deactivated.

If an output is specified that has not been implemented, only the internal memory of the corresponding WY address is set or reset.

### Interrupt operation

The interrupt program will be executed when the elapsed value matches the target value. Any interrupt that has been entered into the Tasks list is automatically enabled.A special interrupt program number is assigned to each channel number.

### Channels used by interrupt programs:

| PLC type | FP0, FP-e | FPΣ, FP0H | FP-X (Relay types), FP-XH | FP-X (Transistor types) | FP0R |
|---|---|---|---|---|---|
| Interrupt0 | Channel0 | Channel0 | Channel0 | Channel0 | Channel0 |
| Interrupt1 | Channel1 | Channel1 | Channel1 | Channel1 | Channel1 |
| Interrupt2 | | | Channel2 | Channel2 | |
| Interrupt3 | Channel2 | Channel2 | Channel3 | Channel3 | Channel2 |
| Interrupt4 | Channel3 | Channel3 | Channel4 | Channel4 | Channel3 |
| Interrupt5 | | | Channel5 | Channel5 | |
| Interrupt6 | | | Channel6 | Channel6 | Channel4 |
| Interrupt7 | | | Channel7 | Channel7 | Channel5 |
| Interrupt8 | | | Channel8 | | |
| Interrupt9 | | | Channel9 | | |

| PLC type | FP0, FP-e | FPΣ, FP0H | FP-X (Relay types), FP-XH | FP-X (Transistor types) | FP0R |
|---|---|---|---|---|---|
| Interrupt10 | | | | | |
| Interrupt11 | | | ChannelA | | |
| Interrupt12 | | | ChannelB | | |

## General programming information

- Select the high-speed counter input for the desired channel in the system registers.
- FP-X, FP0R: When a high-speed counter instruction is executed, the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) for the channel used turns to TRUE. No other high-speed counter instruction using the same channel can be executed as long as the control flag is TRUE.
- FP0, FP-e, FPΣ: The high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- To set a PLC output to FALSE that was previously set to TRUE by this instruction, use an RST or MOVE instruction.
- To cancel execution of an instruction, set bit 3 of the data register storing the high-speed counter control code (**sys_wHscOrPulseControlCode**) to TRUE. The high-speed counter control flag then changes to FALSE. To re-enable execution of the high-speed counter instruction, reset bit 3 to FALSE.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if channel number or values of the data table are outside the permissible range
- if high-speed counter has not been set in the system registers

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if channel number or values of the data table are outside the permissible range
- if high-speed counter has not been set in the system registers

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global Variables

| | Class | Identifier | FP Address | IEC Address | Type | Initial | Comment |
|---|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | out_0 | Y0 | %QX0.0 | BOOL | FALSE | output Y0 of PLC |

### POU header

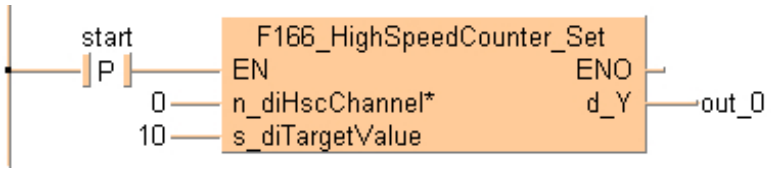
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | out_0 | BOOL | FALSE | output Y0 of PLC |
| 1 | VAR | start | BOOL | FALSE | start condition |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

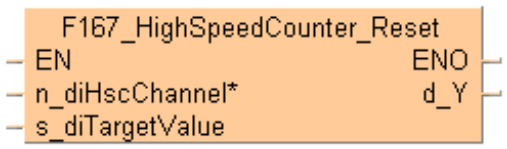
### LD body



# F167_HighSpeedCounter_Reset

Target value match OFF (high-speed counter)

If the elapsed value of the high-speed counter matches the target value, an interrupt process immediately turns the specified output to FALSE.




## Parameters

Input

**n_diHscChannel** (DINT)

High-speed counter channel:
FP-Σ: 0–3
FP-X R: 0–11
FP-X T: 0–7
FP0: 0–3
**F168_PulseOutput_Trapezoidal**: 0–3
**F171_PulseOutput_Trapezoidal**: 0–5

**s_diTargetValue** (DINT)

specify a 32-bit data value for the target value within the following range:
FP0, **F168_PulseOutput_Trapezoidal**: -838808–+8388607
FPΣ, FP-X, **F171_PulseOutput_Trapezoidal**: -2147483467–+2147483648

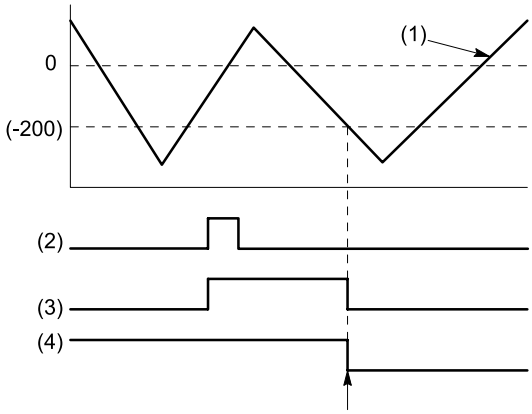
Output

**d_Y** (BOOL)

output which turns to FALSE when the elapsed value matches the target value:
FP-Σ, FP0, **F168_PulseOutput_Trapezoidal**: Y0–Y7
FP-Σ (V3.1 or higher), **F171_PulseOutput_Trapezoidal**: Y0–Y1F
FP-X: Y0–Y29F

## Remarks

Characteristics of target value match OFF control




-200 Target value

(1) Elapsed value of high-speed counter
(2) Execution condition
(3) High-speed counter control flag
(4) PLC output

- The PLC output turns to FALSE when the elapsed value matches the target value. In addition, the high-speed counter control flag turns to FALSE and the instruction is deactivated.
- If an output is specified that has not been implemented, only the internal memory of the corresponding WY address is set or reset.

## Interrupt operation

The interrupt program will be executed when the elapsed value matches the target value. Any interrupt that has been entered into the Tasks list is automatically enabled.A special interrupt program number is assigned to each channel number.

Channels used by interrupt programs:

| PLC type | FP0, FP-e | FPΣ, FP0H | FP-X (Relay types), FP-XH | FP-X (Transistor types) | FP0R |
|---|---|---|---|---|---|
| Interrupt0 | Channel0 | Channel0 | Channel0 | Channel0 | Channel0 |
| Interrupt1 | Channel1 | Channel1 | Channel1 | Channel1 | Channel1 |
| Interrupt2 | | | Channel2 | Channel2 | |
| Interrupt3 | Channel2 | Channel2 | Channel3 | Channel3 | Channel2 |
| Interrupt4 | Channel3 | Channel3 | Channel4 | Channel4 | Channel3 |
| Interrupt5 | | | Channel5 | Channel5 | |
| Interrupt6 | | | Channel6 | Channel6 | Channel4 |
| Interrupt7 | | | Channel7 | Channel7 | Channel5 |
| Interrupt8 | | | Channel8 | | |
| Interrupt9 | | | Channel9 | | |
| Interrupt10 | | | | | |

| PLC type | FP0, FP-e | FPΣ, FP0H | FP-X (Relay types), FP-XH | FP-X (Transistor types) | FP0R |
|---|---|---|---|---|---|
| Interrupt11 | | | ChannelA | | |
| Interrupt12 | | | ChannelB | | |

### General programming information

- Select the high-speed counter input for the desired channel in the system registers.
- FP-X, FP0R: When a high-speed counter instruction is executed, the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) for the channel used turns to TRUE. No other high-speed counter instruction using the same channel can be executed as long as the control flag is TRUE.
- FP0, FP-e, FPΣ: The high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- To set a PLC output to FALSE that was previously set to TRUE by this instruction, use an RST or MOVE instruction.
- To cancel execution of an instruction, set bit 3 of the data register storing the high-speed counter control code (**sys_wHscOrPulseControlCode**) to TRUE. The high-speed counter control flag then changes to FALSE. To re-enable execution of the high-speed counter instruction, reset bit 3 to FALSE.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if channel number or values of the data table are outside the permissible range
- if high-speed counter has not been set in the system registers

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if channel number or values of the data table are outside the permissible range
- if high-speed counter has not been set in the system registers

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global Variables

| | Class | Identifier | FP Address | IEC Address | Type | Initial | Comment |
|---|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | out_0 | Y0 | %QX0.0 | BOOL | FALSE | output Y0 of PLC |

### POU header

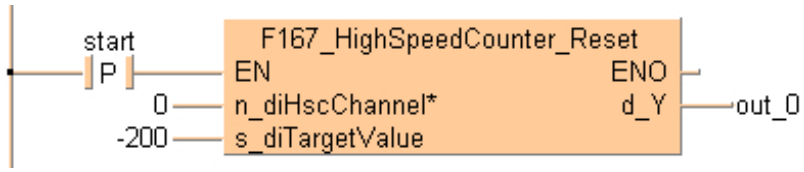
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | out_0 | BOOL | FALSE | output Y0 of PLC |
| 1 | VAR | start | BOOL | FALSE | start condition |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

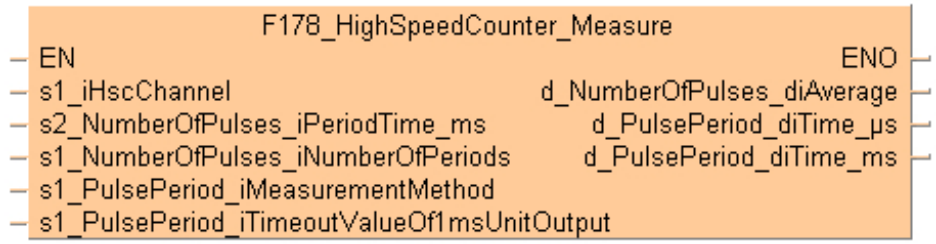
### LD body



# F178_HighSpeedCounter_Measure

Input pulse measurement

This instruction measures the number of input pulses in a specified counting period and the pulse period.




## Parameters

Input

**s1_iHscChannel** (INT)

High-speed counter channel: 0–5

**s2_NumberOfPulses_iPeriodTime_ms** (INT)

Counting period [ms]:
1–5000 (1ms–5s).

**s1_NumberOfPulses_iNumberOfPeriods** (INT)

Number of counting periods: 1–5

**s1_PulsePeriod_iMeasurementMethod** (INT)

Unit of pulse period measurement
0: Pulse period is not measured
1: Pulse period is measured in µ**s**
2: Pulse period is measured in ms
3: Pulse period is measured in µ**s** and ms

**s1_PulsePeriod_iTimeoutValueOf1msUnitOutput** (INT)

Time-out value of pulse period measurement [ms]:
0: no time-out
1: 100ms
2: 200ms
3: 300ms
4: 500ms

Output

**d_NumberOfPulses_diAverage** (DINT)

Average number of pulses per counting period (no. of pulses in counting period/number of counting periods)

**d_PulsePeriod_diTime_μs** (DINT)

Pulse periodμ

**d_PulsePeriod_diTime_ms** (DINT)

Pulse period [ms]

## Characteristics of input pulse measurement

- For input pulse measurement, the channel number, the counting period (1ms–5s) and the number of counting periods (1–5) must be specified. These parameters are used to calculate the average number of input pulses per counting period.
- The unit of pulse period measurement ([μs], [ms] or both) can be specified.
- If the measurement is in μ**s**, the pulse period is measured and output immediately upon execution of this instruction. A maximum of approx. 174.4ms can be measured.
- If the measurement is in ms, the value of the pulse period is updated after every measurement. A maximum of approx. 49.7 days can be measured. A time-out value can be specified after which the measured pulse period is set to -1 if measurement has not been completed.
- During the first counting periods after starting the instruction, the measured pulse period is set to -1 until the specified number of counting periods has been reached.
- If the pulse period is longer than the measurable range or if measurement has not been completed, the measured pulse period is set to -1.

## General programming information

- Select the high-speed counter input for the desired channel in the system registers.
- Keep the execution condition TRUE for pulse measurement using this instruction.
- To stop the measurement, turn the execution condition to FALSE.
- When a high-speed counter instruction is executed, the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) for the channel used turns to TRUE. No other high-speed counter instruction using the same channel can be executed as long as the control flag is TRUE.
- The instruction can be executed simultaneously on a maximum of two channels.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if channel number or values of the data table are outside the permissible range
- if high-speed counter has not been set in the system registers
- if the high-speed counter channel is already used by another high-speed counter or pulse output instruction
- if the number of channels used is 3 or more

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if channel number or values of the data table are outside the permissible range
- if high-speed counter has not been set in the system registers
- if the high-speed counter channel is already used by another high-speed counter or pulse output instruction
- if the number of channels used is 3 or more

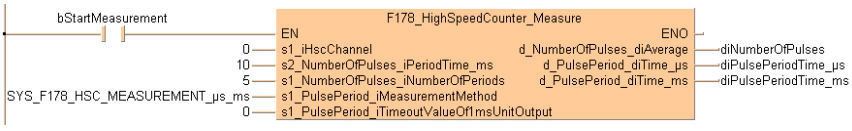
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStartMeasurement | BOOL | FALSE |
| 1 | VAR | diNumberOfPulses | DINT | 0 |
| 2 | VAR | diPulsePeriodTime_µs | DINT | 0 |
| 3 | VAR | diPulsePeriodTime_ms | DINT | 0 |

## LD body



## 20.3 Tool instructions for high-speed counter available for (FPΣ, FPX, FP0R, FPe, FP0)

| **Target value match instructions** | |
|---|---|
| Target value match OFF | Hsc_TargetValueMatch_Reset |
| Target value match ON | Hsc_TargetValueMatch_Set |
| Information and control instructions | |
| Information | Control |
| Evaluating system register settings | |
| HscInfo_IsChannelEnabled | |
| Evaluating and writing special internal flags and special data registers | |
| HscInfo_IsActive | |
| HscInfo_ReadElapsedValue | HscControl_WriteElapsedValue |
| HscInfo_ReadTargetValue | |
| HscInfo_GetCurrentSpeed | |

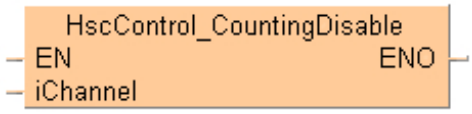
| **Evaluating and writing the control code (DT90052)** | |
|---|---|
| Information | |
| HscInfo_GetControlCode | |
| HscInfo_IsElapsedValueReset | |
| HscInfo_IsCountingDisabled | |
| HscInfo_IsResetInputDisabled | |
| Set | Reset |
| | HscControl_SetDefaults |
| HscControl_ElapsedValueReset | HscControl_ElapsedValueContinue |
| HscControl_CountingDisable | HscControl_CountingEnable |
| HscControl_ResetInputDisable | HscControl_ResetInputEnable |
| HscControl_HscInstructionClear | |

### 20.3.1 High-speed counter control instructions

# HscControl_CountingDisable

Disables counting on high-speed counter channel

This instruction disables counting on the high-speed counter channel specified by **iChannel**. Bit 1 of the Writing the high-speed counter control code (page 1163) is set to TRUE.




## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

## Example

## POU header

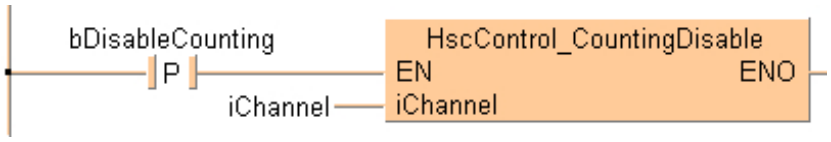
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bDisableCounting | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

## POU body

When the variable **bDisableCounting** changes from FALSE to TRUE, counting on the channel specified by **iChannel** is disabled.

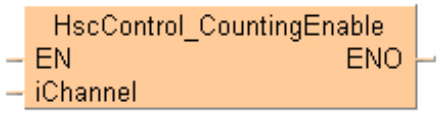
### LD body


bDisableCounting
P
iChannel
HscControl_CountingDisable
EN
ENO
iChannel

# HscControl_CountingEnable

Enables counting on high-speed counter channel

This instruction enables counting on the high-speed counter channel specified by **iChannel** after counting has been disabled with **HscControl_CountingDisable** (page 1197). Bit 1 of the Writing the high-speed counter control code (page 1163) is set to FALSE.




## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

## Example

## POU header

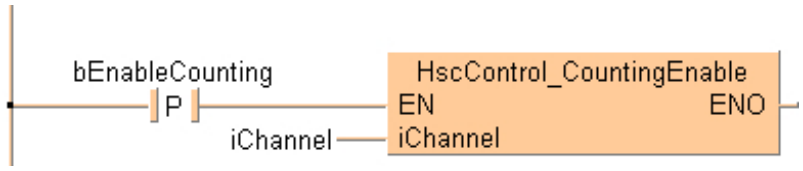
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnableCounting | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

## POU body

When the variable **bEnableCounting** changes from FALSE to TRUE, counting on the channel specified by **iChannel** is enabled.

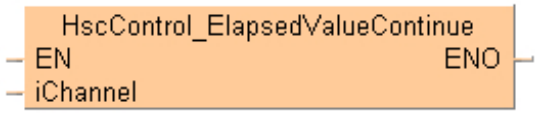
### LD body


bEnableCounting
P
iChannel
HscControl_CountingEnable
EN
ENO
iChannel

# HscControl_ElapsedValueContinue

Continues counting after reset

This instruction resumes counting on the channel specified by **iChannel** after a reset of the elapsed value using **HscControl_ElapsedValueReset** (page 1203). Bit 0 of the Writing the high-speed counter control code (page 1163) is set to FALSE.




## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

## Example

## POU header

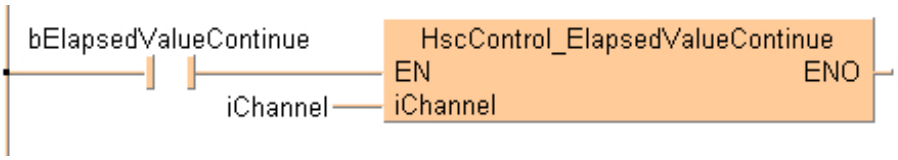
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bElapsedValueContinue | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

## POU body

When the variable **bElapsedValueContinue** is set to TRUE, counting resumes on the channel specified by **iChannel**.

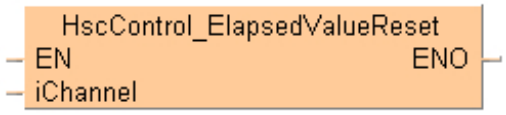
### LD body


bElapsedValueContinue
HscControl_ElapsedValueContinue
EN
ENO
iChannel
iChannel

# HscControl_ElapsedValueReset

Sets elapsed value to 0

This instruction sets the elapsed value of the high-speed counter channel specified by **iChannel** to 0. Bit 0 of the Writing the high-speed counter control code (page 1163) is set to TRUE.




## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

## Example

## POU header

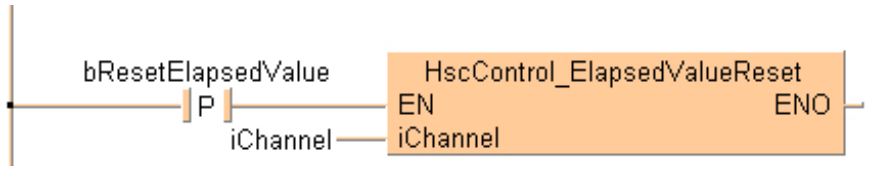
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bResetElapsedValue | BOOL | FALSE |

## POU body

When the variable **bResetElapsedValue** changes from FALSE to TRUE, the elapsed value on the channel specified by **iChannel** is set to 0.

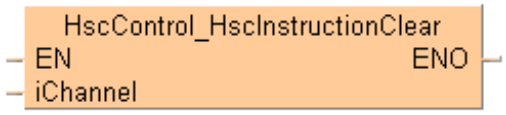
### LD body


bResetElapsedValue
P
iChannel
HscControl_ElapsedValueReset
EN
ENO
iChannel

# HscControl_HscInstructionClear

Clears high-speed counter instruction

This instruction cancels the execution of a high-speed counter instruction on the channel specified by **iChannel**. Bit 3 of the Writing the high-speed counter control code (page 1163) is set to TRUE and subsequently reset to FALSE.




## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

## Example

## POU header

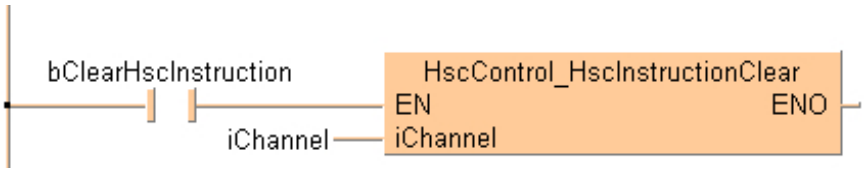
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bClearHscInstruction | BOOL | FALSE |

## POU body

When the variable **bClearHscInstruction** is set to TRUE, the execution of a high-speed counter instruction on the channel specified by **iChannel** is canceled.

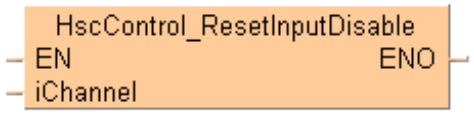
### LD body


bClearHscInstruction
HscControl_HscInstructionClear
EN
ENO
iChannel
iChannel

# HscControl_ResetInputDisable

Disables reset input

This instruction disables the reset input of the high-speed counter channel specified by **iChannel**. Bit 2 of the Writing the high-speed counter control code (page 1163) is set to TRUE.




## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

## Example

## POU header

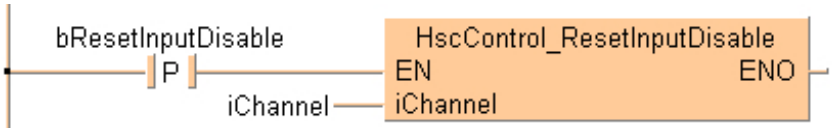
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bResetInputDisable | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

## POU body

When the variable **bResetInputDisable** changes from FALSE to TRUE, the **reset** input of the channel specified by **iChannel** is disabled.

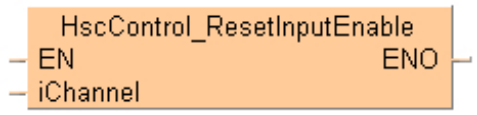
### LD body


bResetInputDisable
P
iChannel
HscControl_ResetInputDisable
EN
ENO
iChannel

# HscControl_ResetInputEnable

Enables reset input

This instruction enables the reset input of the channel specified by **iChannel**. Bit 2 of the Writing the high-speed counter control code (page 1163) is set to FALSE.




## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

## Example

## POU header

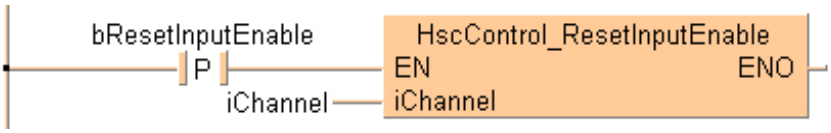
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bResetInputEnable | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

## POU body

When the variable **bResetInputEnable** changes from FALSE to TRUE, the reset input of the channel specified by **iChannel** is enabled.

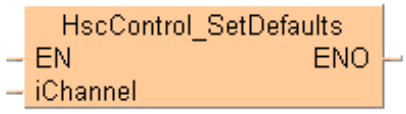
### LD body


bResetInputEnable
P
iChannel
HscControl_ResetInputEnable
EN
ENO
iChannel

# HscControl_SetDefaults

Sets defaults for high-speed counter channel

This instruction sets all bits of the Writing the high-speed counter control code (page 1163) of the channel specified by **iChannel** to 0. 0 is the default setting.




## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

## Example

## POU header

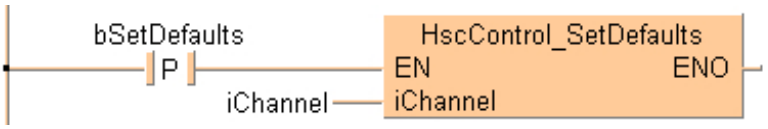
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetDefaults | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

## POU body

When the variable **bSetDefaults** changes from FALSE to TRUE, all settings of the channel specified by **iChannel** are set to their default values.

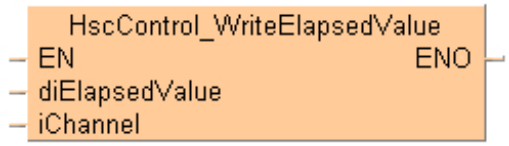
### LD body


bSetDefaults
P
iChannel
HscControl_SetDefaults
EN
ENO
iChannel

# HscControl_WriteElapsedValue

Writes elapsed value into high-speed counter channel

This instruction writes an elapsed value into the high-speed counter channel specified by **iChannel**.




## Parameters

Input

**diElapsedValue** (DINT)

Elapsed value to be written into the channel specified by **iChannel**

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

## Example

## POU header

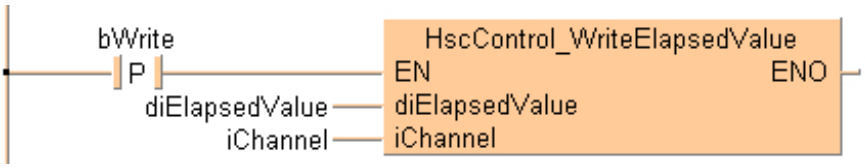
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bWrite | BOOL | FALSE |
| 1 | VAR | diElapsedValue | DINT | 5000 |
| 2 | VAR | iChannel | INT | 0 |

### POU body

When the variable **bWrite** changes from FALSE to TRUE, the elapsed value specified by **diElapsedValue** is written into the channel specified by **iChannel**.

### LD body

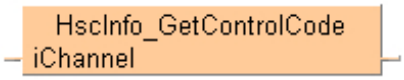



## 20.3.2 High-speed counter information instructions

# HscInfo_GetControlCode

Returns control code of high-speed counter channel

This instruction returns the Writing the high-speed counter control code (page 1163) of the high-speed counter channel specified by **iChannel**.




## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (WORD)

Stores the control code

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | wChannelControlCode | WORD | 0 |

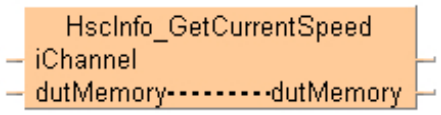
### LD body


HscInfo_GetControlCode
iChannel
iChannel
wChannelControlCode

# HscInfo_GetCurrentSpeed

Returns current speed of high-speed counter channel

This instruction returns the current speed in Hz of the high-speed counter channel specified by **iChannel**.




## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Input/output

**dutMemory** (DUT)

**HscInfo_GetCurrentSpeed_DUT**

Output

**VAR_OUT** (DINT)

Stores the current speed of the channel specified by **iChannel**

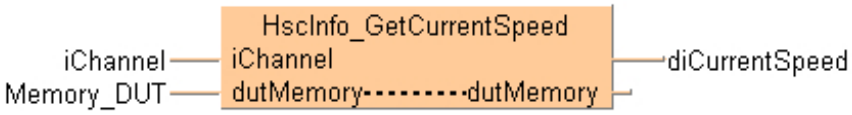
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | Memory_DUT | HscInfo_GetCurrentSpeed_DUT | |
| 2 | VAR | diCurrentSpeed | DINT | 0 |

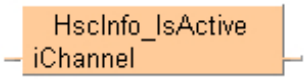
### LD body



# HscInfo_IsActive

Checks if high-speed counter is active

This instruction evaluates the high-speed counter control flags and returns TRUE if the high-speed counter channel specified by **iChannel** is active.




## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (BOOL)

TRUE if the high-speed counter channel specified by **iChannel** is active

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

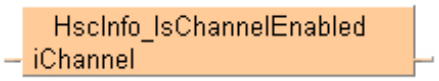
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bChannelActive | BOOL | FALSE |

### LD body

# HscInfo_IsChannelEnabled

Checks if high-speed counter channel is enabled

This instruction returns TRUE if the high-speed counter channel specified by **iChannel** has been enabled in the system registers and is supported by the selected PLC type.




## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (WORD)

TRUE if the channel specified by **iChannel** is enabled

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bChannelEnabled | BOOL | FALSE |

### LD body


HscInfo_IsChannelEnabled
iChannel
iChannel
bChannelEnabled

# HscInfo_IsCountingDisabled

Checks if counting is disabled

This instruction returns TRUE if counting on the channel specified by **iChannel** has been disabled.


## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (BOOL)

TRUE if counting on the channel specified by **iChannel** is disabled

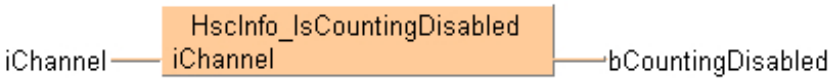
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bCountingDisabled | BOOL | FALSE |

### LD body


HscInfo_IsCountingDisabled
iChannel
iChannel
bCountingDisabled

# HscInfo_IsElapsedValueReset

Checks if elapsed value is set to 0

This instruction returns TRUE if the elapsed value of the high-speed counter channel specified by **iChannel** has been reset to 0.


## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (BOOL)

TRUE if the channel specified by **iChannel** has been reset

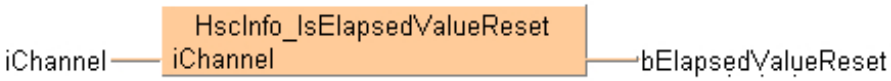
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bElapsedValueReset | BOOL | FALSE |

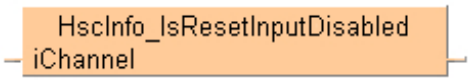
### LD body


HscInfo_IsElapsedValueReset
iChannel
iChannel
bElapsedValueReset

# HscInfo_IsResetInputDisabled

Checks if reset input is disabled

This instruction returns TRUE if the reset input of the channel specified by **iChannel** has been disabled.




## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (WORD)

TRUE if the reset input of the channel specified by **iChannel** is disabled

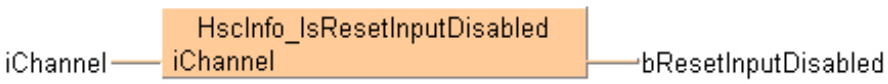
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bResetInputDisabled | BOOL | FALSE |

### LD body


HscInfo_IsResetInputDisabled
iChannel
iChannel
bResetInputDisabled

# HscInfo_ReadElapsedValue

Reads elapsed value from high-speed counter channel

This instruction reads the elapsed value from the high-speed counter channel specified by **iChannel**.


## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (DINT)

Stores the elapsed value from the channel specified by **iChannel**

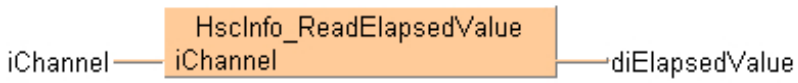
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | diElapsedValue | DINT | 0 |

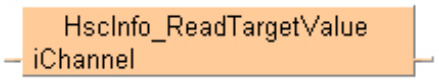
### LD body


HscInfo_ReadElapsedValue
iChannel
iChannel
diElapsedValue

# HscInfo_ReadTargetValue

Reads target value from high-speed counter channel

This instruction reads the target value from the high-speed counter channel specified by **iChannel**.




## Parameters

Input

**iChannel** (INT)

High-speed counter channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (DINT)

Stores the target value of the channel specified by **iChannel**

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | diTargetValue | DINT | 0 |

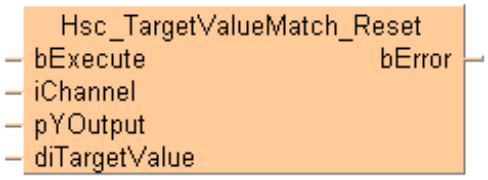
### LD body


## 20.3.3 High-speed counter target value match control

# Hsc_TargetValueMatch_Reset

Target value match OFF (high-speed counter)

If the elapsed value matches the target value **diTargetValue** of the high-speed counter channel specified by **iChannel**, the output relay specified by **pYOutput** immediately turns to FALSE.




## Parameters

Input

**bExecute** (BOOL)

A rising edge activates the function; evaluate the high-speed counter control flag using HscInfo_IsActive

**iChannel** (INT)

FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

**pYOutputPOINTER**

Pointer result obtained by GetPointer from a global variable that supplies the channel number and output

**diTargetValue** (DINT)

Specify a 32-bit data value for the target value within the following range:
FP0, FP-e: -838808–+8388607
FPΣ, FP-X, FP0R: -2147483467–+2147483648

Output

**iError** (BOOL)

TRUE if the combination of `Channel%d` and `pYOuput.iOffset` does not match a valid combination of channel number and output as determined by the global variable

## Remarks

This non-inline instruction is part of the tool instructions for high-speed counters. For a detailed description of the instruction(s) used internally, please refer to the online help: F167_HighSpeedCounter_Reset

Note

To validate the combination of channel and Y output, the compiler requires the following name pattern for global variables:`%sHsc_TargetValueMatch_Channel%d_Y%d_%s`

Always use this pattern for global variables in target value match control.

- `Channel%d` must be a high-speed counter channel number enabled in the system registers
- `Y%d` must be an explicit output address supported by the PLC

| | |
|---|---|
| FP-Σ, FP0, FP-e: | Y0–Y7 |
| FP-Σ (V3.1 or higher), FP0R: | Y0–Y1F |
| FP-X: | Y0–Y29F |

- `%s` is an optional descriptor at the beginning of the pattern
- `_%s` is an optional descriptor at the end of the pattern

### Example

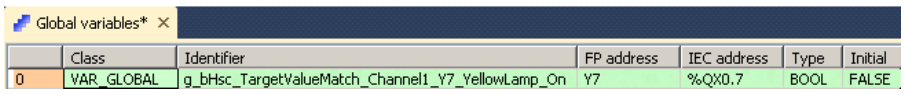
| | | |
|---|---|---|
| Optional | Fixed pattern | Optional |
| `g_b` | `Hsc_TargetValueMatch_Channel1A_Y11F` | `_MotorOn` |

This global variable generates the code for channel **A** and output **Y11F**.

## Example

## Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.



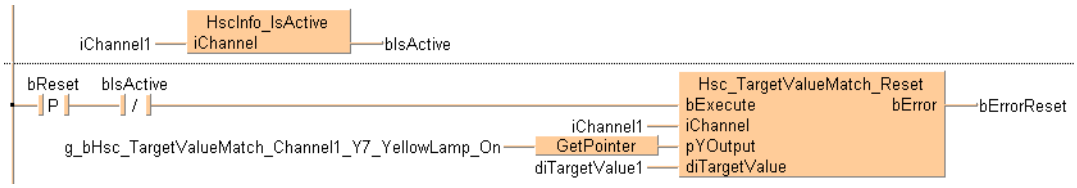

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_bHsc_TargetValueMatch_Channel1_Y7_YellowLamp_On | BOOL | FALSE |
| 1 | VAR | iChannel1 | INT | 1 |
| 2 | VAR | diTargetValue1 | DINT | 25000 |
| 3 | VAR | bError | BOOL | TRUE |
| 4 | VAR | bIsActive | BOOL | FALSE |
| 5 | VAR | bReset | BOOL | FALSE |
| 6 | VAR | bErrorReset | BOOL | FALSE |

## LD body

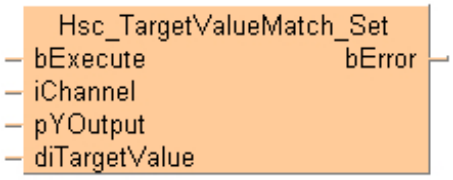
Use **HscInfo_IsActive** to evaluate the channel specified by **iChannel1**. If a rising edge is detected at **bReset** and if **bIsActive** is not TRUE, the instruction is executed. The combination of channel number and output contact is validated in the global variable **g_bHsc_TargetValueMatch_Channel1_Y7_YellowLamp_On**. When the high-speed counter on channel 1 reaches the target value **diTargetValue1**, output Y7 is set to FALSE.



# Hsc_TargetValueMatch_Set

Target value match ON (high-speed couter)

If the elapsed value matches the target value **diTargetValue** of the high-speed counter channel specified by **iChannel**, the output relay specified by **pYOutput** immediately turns to TRUE.




## Parameters

Input

**bExecute**

A rising edge activates the function; evaluate the high-speed counter control flag using HscInfo_IsActive

**iChannel** (INT)

FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

**pYOutputPOINTER**

Pointer result obtained by GetPointer from a global variable that supplies the channel number and output

**diTargetValue** (DINT)

Specify a 32-bit data value for the target value within the following range:
FP0, FP-e: -838808–+8388607
FPΣ, FP-X, FP0R: -2147483467–+2147483648

Output

**bError** (BOOL)

TRUE if the combination of `Channel%d` and `pYOuput.iOffset` does not match a valid combination of channel number and output as determined by the global variable

## Remarks

This non-inline instruction is part of the tool instructions for high-speed counters. For a detailed description of the instruction(s) used internally, please refer to the online help: F167_HighSpeedCounter_Reset **F166_HighSpeedCounter_Set**

Note

To validate the combination of channel and Y output, the compiler requires the following name pattern for global variables:`%sHsc_TargetValueMatch_Channel%d_Y%d_%s`

Always use this pattern for global variables in target value match control.

- `Channel%d` must be a high-speed counter channel number enabled in the system registers
- `Y%d` must be an explicit output address supported by the PLC

| | |
|---|---|
| FP-Σ, FP0, FP-e: | Y0–Y7 |
| FP-Σ (V3.1 or higher), FP0R: | Y0–Y1F |
| FP-X: | Y0–Y29F |

- `%s` is an optional descriptor at the beginning of the pattern
- `_%s` is an optional descriptor at the end of the pattern

### Example

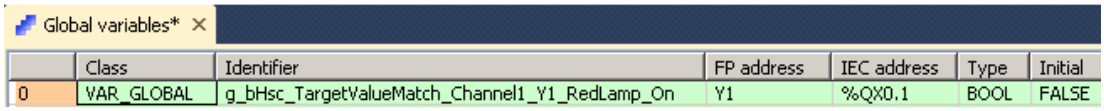
| Optional | Fixed pattern | Optional |
|---|---|---|
| `g_b` | `Hsc_TargetValueMatch_ChannelA_Y11F` | `_MotorOn` |

This global variable generates the code for channel **A** and output **Y11F**.

## Example

## Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.



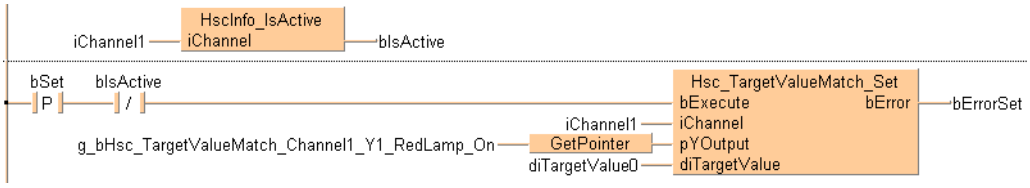

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_bHsc_TargetValueMatch_Channel1_Y1_RedLamp_On | BOOL | FALSE |
| 1 | VAR | diTargetValue0 | DINT | 11000 |
| 2 | VAR | iChannel1 | INT | 1 |
| 3 | VAR | bIsActive | BOOL | FALSE |
| 4 | VAR | bErrorSet | BOOL | FALSE |
| 5 | VAR | bSet | BOOL | FALSE |

## LD body

Use **HscInfo_IsActive** to evaluate the channel specified by **iChannel1**. If a rising edge is detected at **bSet** and if **bIsActive** is not TRUE, the instruction is executed. The combination of channel number and output contact is validated in the global variable **g_bHsc_TargetValueMatch_Channel1_Y1_RedLamp_On**. When the high-speed counter on channel 1 reaches the target value **diTargetValue0**, output Y1 is set to TRUE.

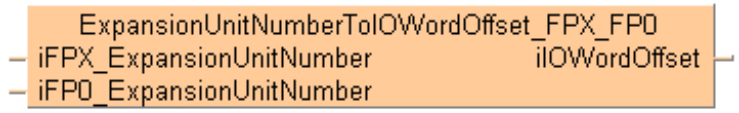


# 21 Input, output and unit access instructions

# ExpansionUnitNumberToIOWordOffset_FPX_FP0

Function to calculate the I/O offset of an analog expansion unit connected to the CPU via an adapter.

I/O starting addresses vary, depending on the installation positions of the adapter and of the expansion unit. The function calculates the I/O word offset based on the adapter position and the unit position relative to the adapter.




## Parameters

Input

**iFPX_ExpansionUnitNumber** (INT)

Set the position of the adapter relative to the CPU.
Values: 1 to 8

**iFP0_ExpansionUnitNumber** (INT)

Set the position of the FP0/FP0R expansion unit relative to the adapter.
Values: 1 to 3

Output

**iIOWordOffset** (INT)

Returns the offset of the I/O word (WX/WY)

## Example

### POU header

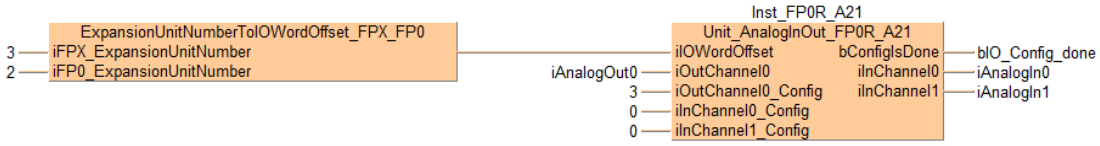
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | Inst_FP0R_A21 | Unit_AnalogInOut_FP0R_A21 | |
| 2 | VAR | iAnalogOut0 | INT | 0 |
| 3 | VAR | bIO_Config_done | BOOL | FALSE |
| 4 | VAR | iAnalogIn0 | INT | 0 |
| 5 | VAR | iAnalogIn1 | INT | 0 |

## POU body

The function calculates the I/O word offset for an FP0/FP0R analog unit connected to an FP-X CPU. **iFPX_ExpansionUnitNumber** is the installation position (1 to 8) of the FP0/FP0R expansion adapter. **iFP0_ExpansionUnitNumber** is the installation position of the analog unit relative to the adapter (1 to 3).

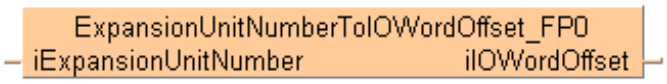
## LD body



# ExpansionUnitNumberToIOWordOffset_FP0

Function to calculate the I/O offset of an analog expansion unit connected directly to the CPU (without adapter).

I/O starting addresses vary, depending on the installation position of the expansion unit. The function block calculates the I/O word offset based on the unit position relative to the CPU.




## Parameters

Input

**iExpansionUnitNumber** (INT)

Set the position of the FP0/FP0R expansion unit relative to the CPU.
Values: 1 to 3

Output

**iIOWordOffset** (INT)

Returns the offset of the I/O word (WX/WY2, 4, or 6)

## Example

### POU header

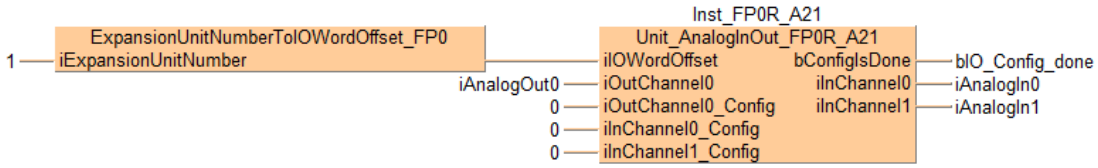
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | Inst_FP0R_A21 | Unit_AnalogInOut_FP0R_A21 | |
| 2 | VAR | iAnalogOut0 | INT | 0 |
| 3 | VAR | bIO_Config_done | BOOL | FALSE |
| 4 | VAR | iAnalogIn0 | INT | 0 |
| 5 | VAR | iAnalogIn1 | INT | 0 |

### POU body

The function converts the position number (1 to 3) of the FP0/FP0R expansion unit, where 0 is the CPU, to the corresponding I/O word offset 2, 4, or 6.

## LD body


Inst_FP0R_A21
ExpansionUnitNumberToIOWordOffset_FP0
Unit_AnalogInOut_FP0R_A21
1
iExpansionUnitNumber
iIOWordOffset
bConfigIsDone
bIO_Config_done
iAnalogOut0
iOutChannel0
iInChannel0
iAnalogIn0
0
iOutChannel0_Config
iInChannel1
iAnalogIn1
0
iInChannel0_Config
0
iInChannel1_Config

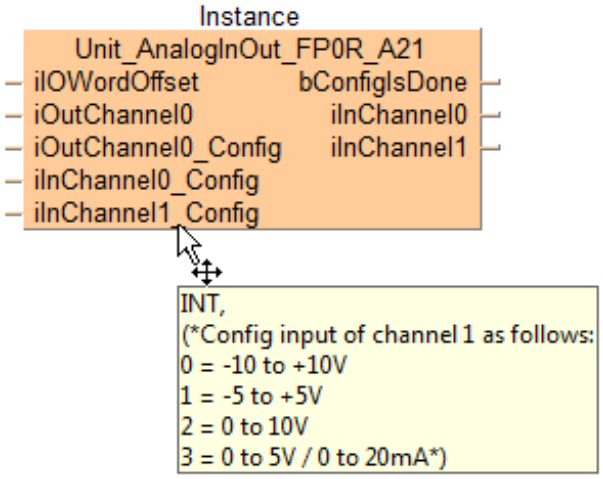
# Unit_AnalogInOut_FP0R_A21

Function block to write to and read from an FP0R-A21 unit.

This function block writes digital data to the analog output channel of the analog unit and reads converted digital data from its analog input channels. The digital values to be converted and output as analog values are specified at **iOutChannel0**. The converted digital values from the analog unit are stored per channel in the output variables **iInChannel0** and **iInChannel1**.

The analog output and input ranges are also set with this function block.

The voltage or current output must be set with the DIP switches.




## Parameters

Input

**iIOWordOffset** (INT)

Set the offset of the first WX/WY address of the analog unit according to its installation position.

For analog expansion units connected directly to the CPU (without adapter): Use **ExpansionUnitToIOWordOffset_FP0** or make the following settings: 2 (WX2/WY2) for unit number 1, 4 (WX4/WY4) for unit number 2, 6 (WX6/WY6) for unit number 3

For analog expansion units connected to the CPU via an adapter: Use **ExpansionUnitToIOWordOffset_FPX_FP0** or select the offset from the table.

| Unit position relative to the adapter | Adapter position relative to the CPU | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1st unit | 2nd unit | 3rd unit | 4th unit | 5th unit | 6th unit | 7th unit | 8th unit |
| 1st unit | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 2nd unit | 32 | 42 | 52 | 62 | 72 | 82 | 92 | 102 |
| 3rd unit | 34 | 44 | 54 | 64 | 74 | 84 | 94 | 104 |

**iOutChannel0** (INT)

Set the digital value to be converted and output by the analog unit.
Values:
For -10 to +10V, -5 to +5V: -8000 to +8000
For 0 to 10V, 0 to 5 V, 0 to 20mA, 4 to 20mA: 0 to 16000

**iOutChannel0_Config** (INT)

Set the voltage or current range for the analog output channel.
Values:
0: -10 to +10V, 0 to 20mA
1: -5 to +5V, 4 to 20mA
2: 0 to 10V
3: 0 to 5V

**iInChannel0_Config**, **iInChannel1_Config** (INT)

Set the voltage or current range for the analog input channel.
Values:
0: -10 to +10V
1: -5 to +5V
2: 0 to 10V
3: 0 to 5V, 0 to 20mA (depending on wiring method)

Output

**bConfigIsDone** (BOOL)

TRUE when I/O configuration is completed and the unit is ready.

**iInChannel0**, **iInChannel1** (INT)

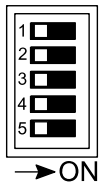
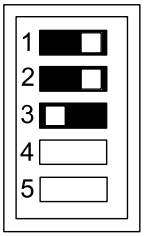
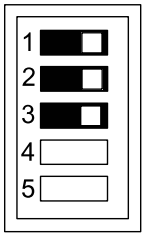
Returns the converted digital data from the analog unit by channel.
Values:
For -10 to +10V, -5 to +5V: -8000 to +8000
For 0 to 10V, 0 to 5 V, 0 to 20mA, 4 to 20mA: 0 to 16000

## DIP switch settings

DIP switches 1 and 2 must be ON to use 14-bit mode. DIP switch 3 is used to set voltage or current output, and DIP switch 5 is used to turn averaging on or off. DIP switch 4 is not used and can be either ON or OFF.

The DIP switch settings will become effective when the power is turned from OFF to ON.

**Voltage or current output:**

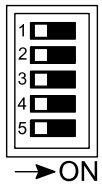
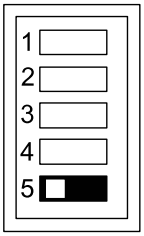
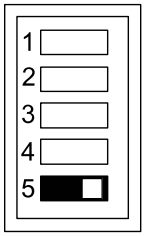
| | Voltage output | Current output |
|---|---|---|
|  |  |  |



**Input averaging:**

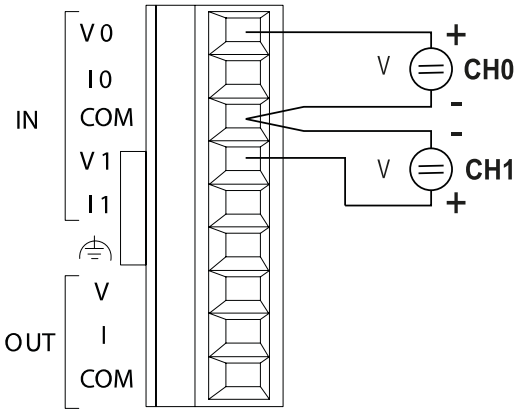
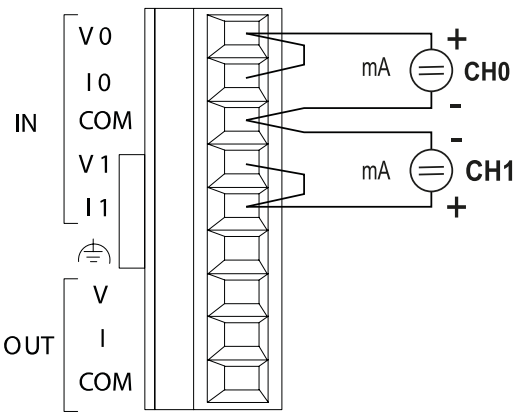
No averaging: Conversion data is set for the specified input contact point area for each A/D conversion, on each channel.

Averaging: On each channel, for each A/D conversion, the maximum and minimum values from the data of the last ten times are excluded, the data from the other eight times is averaged and the result is set.

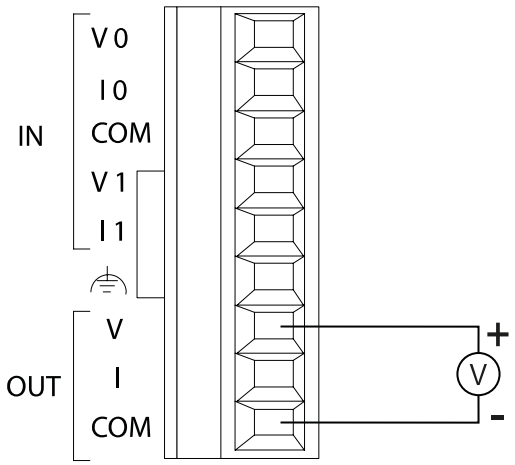
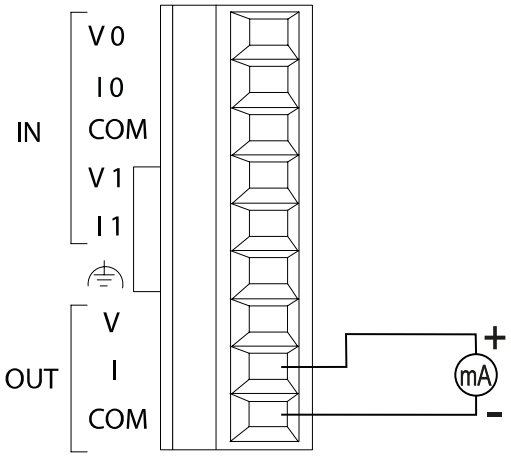
| | No averaging | Averaging |
|---|---|---|
|  |  |  |



## Input wiring

| Voltage input | Current input |
|---|---|
|  |  |
| Connect the input instrument between the IN/V and IN/COM terminals. | Connect the IN/V and IN/I terminals. Connect the input instrument between the bridge and the IN/COM terminal. |

## Output wiring

| Voltage output | Current output |
|---|---|
|  |  |
| Connect the output instrument between the OUT/V and OUT/COM terminals. | Connect the output instrument between the OUT/I and OUT/COM terminals. |

## Conversion characteristics

| -10V to +10V DC input or output | | -5V to +5V DC input or output | | 0V to 5V DC input or output | |
|---|---|---|---|---|---|
| **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** |
| -8000 | -10V | -8000 | -5V | 0 | 0.0V |
| -4000 | -5V | -4000 | -2.5V | 4000 | 1.25V |
| 0 | 0V | 0 | 0V | 8000 | 2.5V |
| +4000 | +5V | +4000 | +2.5V | 12000 | 3.75V |
| +8000 | +10V | +8000 | +5V | 16000 | 5.0V |

| 0V to 10V DC input or output | | 0mA to 20mA input or output | | 4mA to 20mA output | |
|---|---|---|---|---|---|
| **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** |
| 0 | 0.0V | 0 | 0.0mA | 0 | 4.0mA |
| 4000 | 2.5V | 3200 | 4.0mA | 4000 | 8.0mA |
| 8000 | 5.0V | 6400 | 8.0mA | 8000 | 12.0mA |
| 12000 | 7.5V | 9600 | 12.0mA | 12000 | 16.0mA |
| 16000 | 10.0V | 12800 | 16.0mA | 16000 | 20.0mA |
| | | 16000 | 20.0mA | | |

Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP0R Analog I/O Unit User's Manual”

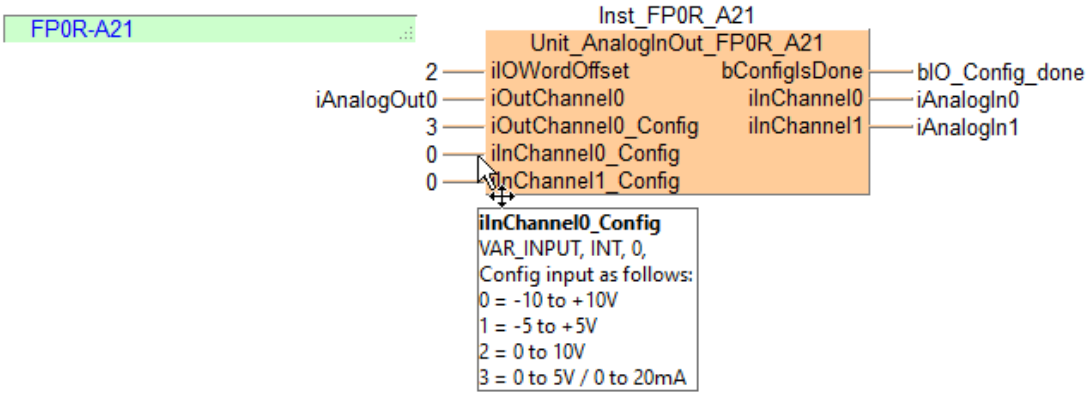
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | Inst_FP0R_A21 | Unit_AnalogInOut_FP0R_A21 | |
| 2 | VAR | iAnalogOut0 | INT | 0 |
| 3 | VAR | bIO_Config_done | BOOL | FALSE |
| 4 | VAR | iAnalogIn0 | INT | 0 |
| 5 | VAR | iAnalogIn1 | INT | 0 |

### LD body



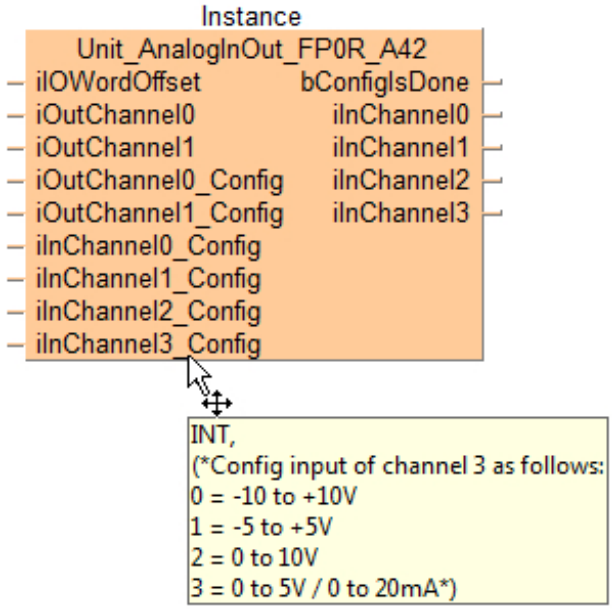
# Unit_AnalogInOut_FP0R_A42

Function block to write to and read from an FP0R-A42 unit.

This function writes digital data to the analog output channels of the analog unit and reads converted digital data from its analog input channels. The digital values to be converted and output as analog values are specified at **iOutChannel0** and **iOutChannel1**. The converted digital values from the analog unit are stored per channel in the output variables **iInChannel0** to **iInChannel3**.

The analog output and input ranges are also set with this function block.

The voltage or current output must be set with the DIP switches.




## Parameters

Input

**iIOWordOffset** (INT)

Set the offset of the first WX/WY address of the analog unit according to its installation position.

For analog expansion units connected directly to the CPU (without adapter): Use **ExpansionUnitToIOWordOffset_FP0** or make the following settings: 2 (WX2/WY2) for unit number 1, 4 (WX4/WY4) for unit number 2, 6 (WX6/WY6) for unit number 3

For analog expansion units connected to the CPU via an adapter: Use **ExpansionUnitToIOWordOffset_FPX_FP0** or select the offset from the table.

| Unit position relative to the adapter | Adapter position relative to the CPU | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1st unit | 2nd unit | 3rd unit | 4th unit | 5th unit | 6th unit | 7th unit | 8th unit |
| 1st unit | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 2nd unit | 32 | 42 | 52 | 62 | 72 | 82 | 92 | 102 |
| 3rd unit | 34 | 44 | 54 | 64 | 74 | 84 | 94 | 104 |

**iOutChannel0**, **iOutChannel1** (INT)

Set the digital value to be converted and output by the analog unit.
Values:
For -10 to +10V, -5 to +5V: -8000 to +8000
For 0 to 10V, 0 to 5 V, 0 to 20mA, 4 to 20mA: 0 to 16000

**iOutChannel0_Config**, **iOutChannel1_Config** (INT)

Set the voltage or current range for the analog output channel.
Values:
0: -10 to +10V, 0 to 20mA
1: -5 to +5V, 4 to 20mA
2: 0 to 10V
3: 0 to 5V

**iInChannel0_Config**, **iInChannel1_Config**, **iInChannel2_Config**, **iInChannel3_Config** (INT)

Set the voltage or current range for the analog input channel.
Values:
0: -10 to +10V
1: -5 to +5V
2: 0 to 10V
3: 0 to 5V, 0 to 20mA (depending on wiring method)

Output

**bConfigIsDone** (BOOL)

TRUE when I/O configuration is completed and the unit is ready.

**iInChannel0** to **iInChannel3** (INT)

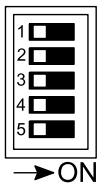
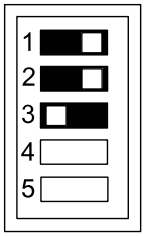
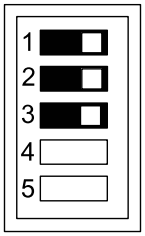
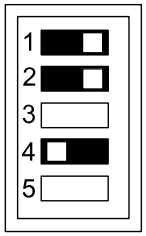
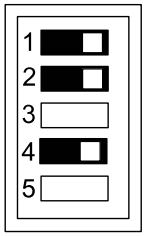
Returns the converted digital data from the analog unit by channel.
Values:
For -10 to +10V, -5 to +5V: -8000 to +8000
For 0 to 10V, 0 to 5 V, 0 to 20mA, 4 to 20mA: 0 to 16000

## DIP switch settings

DIP switches 1 and 2 must be ON to use 14-bit mode. DIP switch 3 is used to set voltage or current output for channel 0, DIP switch 4 is used to set voltage or current output for channel 1, and DIP switch 5 is used to turn averaging on or off.

The DIP switch settings will become effective when the power is turned from OFF to ON.

**Voltage or current output:**

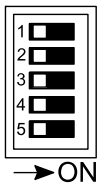
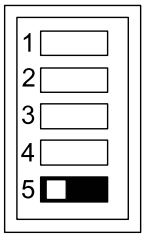
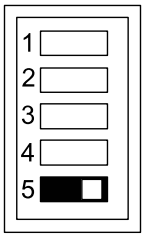
| | Channel 0 | | Channel 1 | |
|---|---|---|---|---|
| | Voltage output | Current output | Voltage output | Current output |
|  |  |  |  |  |



**Input averaging:**

No averaging: Conversion data is set for the specified input contact point area for each A/D conversion, on each channel.

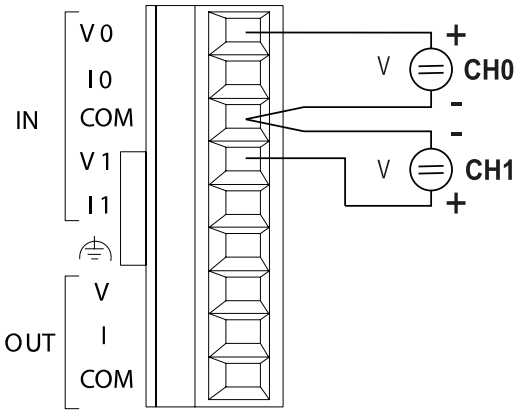
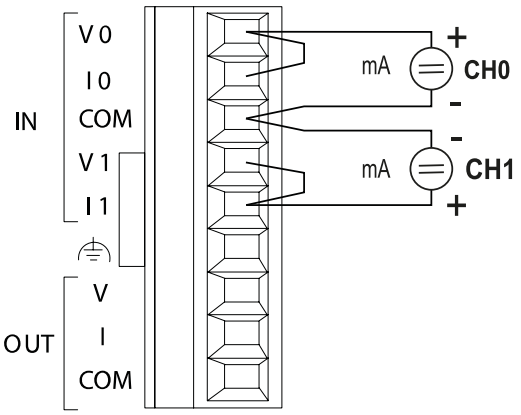
Averaging: On each channel, for each A/D conversion, the maximum and minimum values from the data of the last ten times are excluded, the data from the other eight times is averaged and the result is set.

| | No averaging | Averaging |
|---|---|---|
|  |  |  |

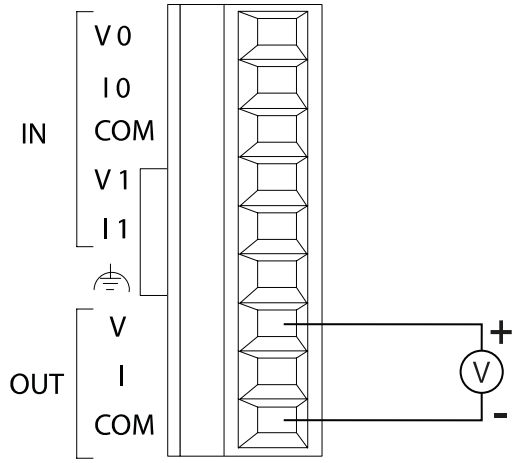
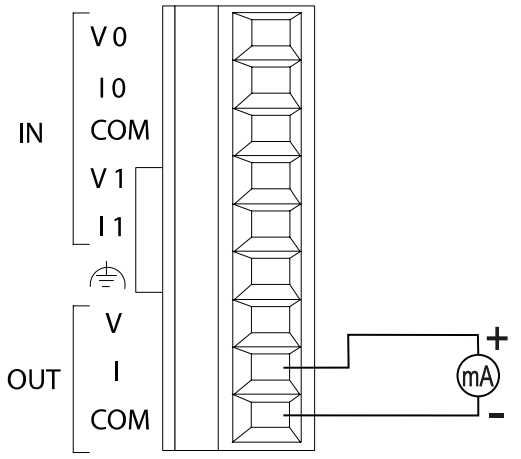


## Input wiring

The examples show the wiring for input channels 0 and 1 and output channel 0.

| Voltage input | Current input |
|---|---|
|  |  |
| Connect the input instrument between the IN/V and IN/COM terminals. | Connect the IN/V and IN/I terminals. Connect the input instrument between the bridge and the IN/COM terminal. |

## Output wiring

| Voltage output | Current output |
|---|---|
|  |  |
| Connect the output instrument between the OUT/V and OUT/COM terminals. | Connect the output instrument between the OUT/I and OUT/COM terminals. |

## Conversion characteristics

| -10V to +10V DC input or output | | -5V to +5V DC input or output | | 0V to 5V DC input or output | |
|---|---|---|---|---|---|
| **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** |
| -8000 | -10V | -8000 | -5V | 0 | 0.0V |
| -4000 | -5V | -4000 | -2.5V | 4000 | 1.25V |
| 0 | 0V | 0 | 0V | 8000 | 2.5V |

| -10V to +10V DC input or output | | -5V to +5V DC input or output | | 0V to 5V DC input or output | |
|---|---|---|---|---|---|
| **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** |
| +4000 | +5V | +4000 | +2.5V | 12000 | 3.75V |
| +8000 | +10V | +8000 | +5V | 16000 | 5.0V |

| 0V to 10V DC input or output | | 0mA to 20mA input or output | | 4mA to 20mA output | |
|---|---|---|---|---|---|
| **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** |
| 0 | 0.0V | 0 | 0.0mA | 0 | 4.0mA |
| 4000 | 2.5V | 3200 | 4.0mA | 4000 | 8.0mA |
| 8000 | 5.0V | 6400 | 8.0mA | 8000 | 12.0mA |
| 12000 | 7.5V | 9600 | 12.0mA | 12000 | 16.0mA |
| 16000 | 10.0V | 12800 | 16.0mA | 16000 | 20.0mA |
| | | 16000 | 20.0mA | | |

Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP0R Analog I/O Unit User's Manual”

## Example

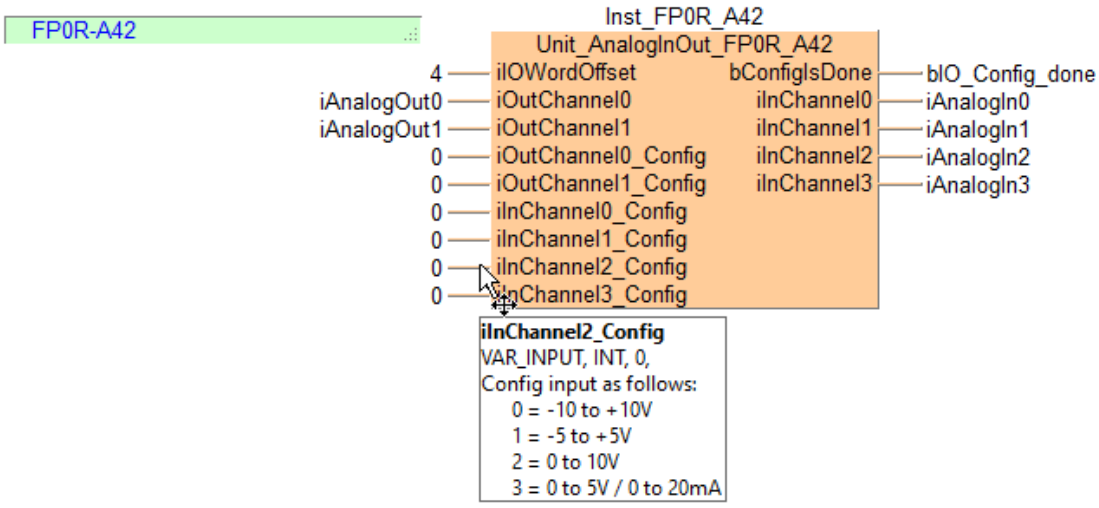
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | Inst_FP0R_A42 | Unit_AnalogInOut_FP0R_A42 | |
| 2 | VAR | iAnalogOut0 | INT | 0 |
| 3 | VAR | iAnalogOut1 | INT | 0 |
| 4 | VAR | bIO_Config_done | BOOL | FALSE |
| 5 | VAR | iAnalogIn0 | INT | 0 |
| 6 | VAR | iAnalogIn1 | INT | 0 |
| 7 | VAR | iAnalogIn2 | INT | 0 |
| 8 | VAR | iAnalogIn3 | INT | 0 |

## LD body

Use **ExpansionUnitNumberToIOWordOffset_FP0** or **ExpansionUnitNumberToIOWordOffset_FPX_FP0** to calculate the word offset of the analog unit connected to the CPU.



# Unit_AnalogInOut_FP0_A21

Function to write to and read from an FP0-A21 unit.

This function writes digital data to the analog output channel of the analog unit and reads converted digital data from its analog input channels. The digital values to be converted and output as analog values are specified at **iOutChannel**. The converted digital values from the analog unit are stored per channel in the output variables **iInChannel0** and **iInChannel1**.

The analog value ranges must be set with the DIP switches.


## Parameters

Input

**iIOWordOffset** (INT)

Set the offset of the first WX/WY address of the analog unit according to its installation position.
For analog expansion units connected directly to the CPU (without adapter): Use **ExpansionUnitToIOWordOffset_FP0** or make the following settings: 2 (WX2/WY2) for unit number 1, 4 (WX4/WY4) for unit number 2, 6 (WX6/WY6) for unit number 3
For analog expansion units connected to the CPU via an adapter: Use **ExpansionUnitToIOWordOffset_FPX_FP0** or select the offset from the table.

| Unit position relative to the adapter | Adapter position relative to the CPU | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1st unit | 2nd unit | 3rd unit | 4th unit | 5th unit | 6th unit | 7th unit | 8th unit |
| 1st unit | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 2nd unit | 32 | 42 | 52 | 62 | 72 | 82 | 92 | 102 |
| 3rd unit | 34 | 44 | 54 | 64 | 74 | 84 | 94 | 104 |

**iOutChannel** (INT)

Set the digital value to be converted and output by the analog unit.
Values:
For -10.0 to +10.0V: -2000 to +2000
For 0.0 to 20.0mA: 0 to 4000

Output

**iInChannel0**, **iInChannel1** (INT)

Returns the converted digital data from the analog unit by channel.

Values:

For 0 to 5V, 0 to 20mA: 0 to 4000 (depending on wiring method)

For -10 to +10V, -100 to 100mV: -2000 to +2000

For 25 to 1000°C: 25 to 1000

For 0 to -100°C: 0 to -100

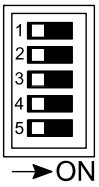
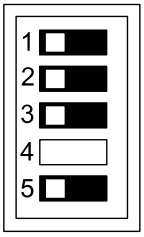
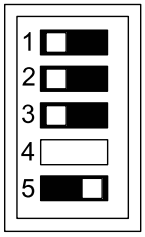
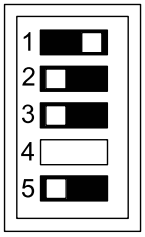
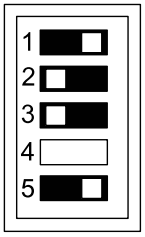
## DIP switch settings

The DIP switch settings will become effective when the power is turned from OFF to ON.

**Input range and averaging:**

No averaging: Conversion data is set for the specified input contact point area for each A/D conversion, on each channel.

Averaging: On each channel, for each A/D conversion, the maximum and minimum values from the data of the last ten times are excluded, the data from the other eight times is averaged and the result is set.

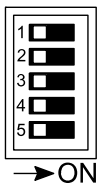
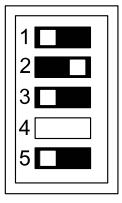
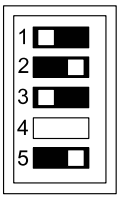
| | 0 to 5V, 0 to 20mA | | -10 to +10V | |
|---|---|---|---|---|
| | No averaging | Averaging | No averaging | Averaging |
|  |  |  |  |  |



**Thermocouple input range:**

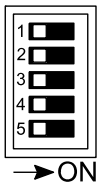
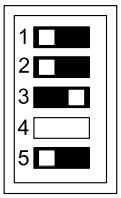
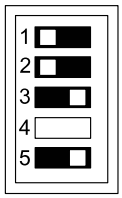
For thermocouples, averaging is always performed.

- Thermocouple type K

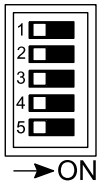
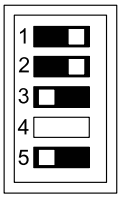
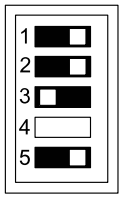
| | Terminal temperature to 1000°C | -100°C to terminal temperature |
|---|---|---|
|  |  |  |



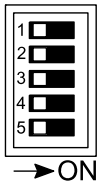
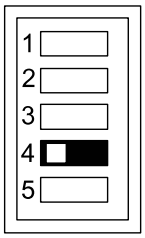
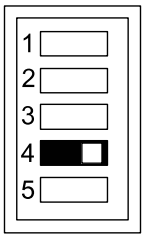
- Thermocouple type J

| | Terminal temperature to 750°C | -100°C to terminal temperature |
|---|---|---|
|   |   |   |

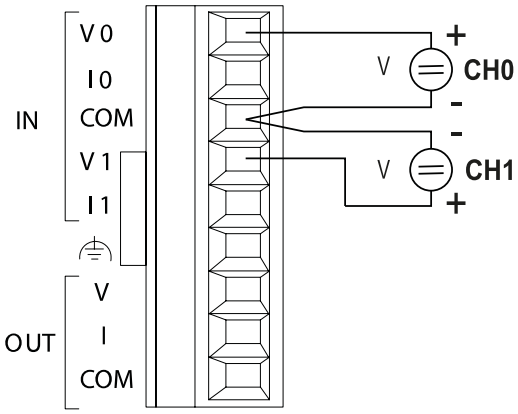
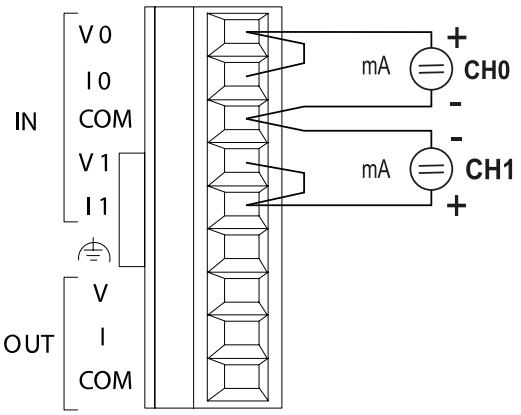
- Thermocouple type T

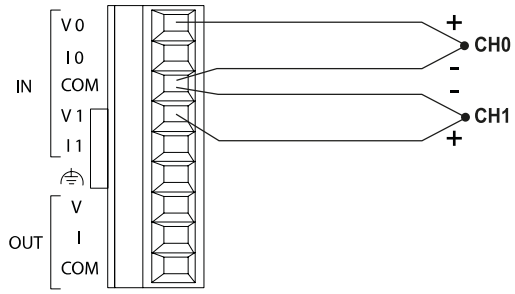
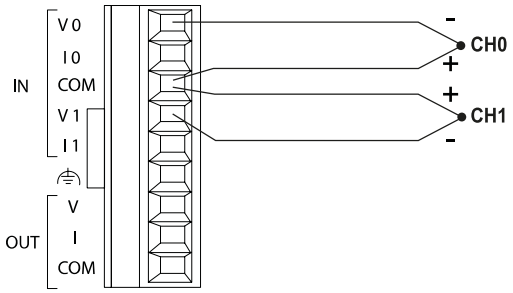
| | Terminal temperature to 350°C | -100°C to terminal temperature |
|---|---|---|
|   |   |   |

**Output range:**

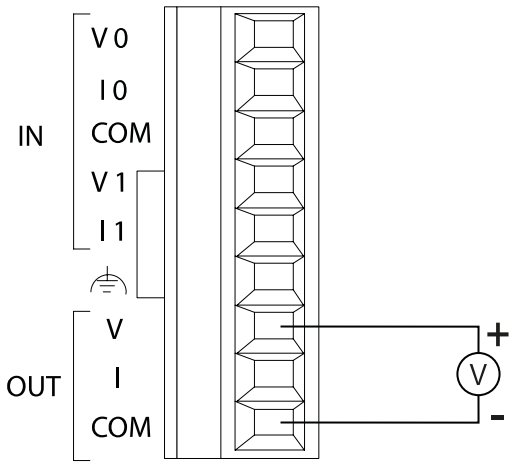
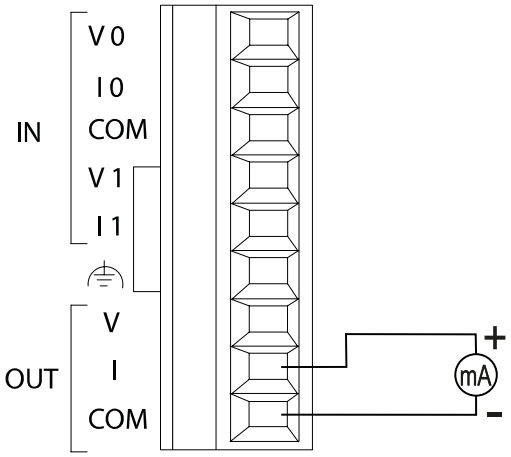
| | 0 to 20mA | -10 to +10V |
|---|---|---|
|   |   |   |

## Input wiring

| Voltage input | Current input |
|---|---|
|  |  |
| Connect the input instrument between the IN/V and IN/COM terminals. | Connect the IN/V and IN/I terminals. Connect the input instrument between the bridge and the IN/COM terminal. |

| Thermocouple input | |
|---|---|
| Temperature higher than terminal temperature: | Temperature lower than terminal temperature: |
|  |  |
| Connect the IN/V terminal to the (+) side of the thermocouple. Connect the IN/COM terminal to the (-) side of the thermocouple. | Connect the IN/V terminal to the (-) side of the thermocouple. Connect the IN/COM terminal to the (+) side of the thermocouple. |

## Output wiring

| Voltage output | Current output |
|---|---|
|  |  |
| Connect the output instrument between the OUT/V and OUT/COM terminals. | Connect the output instrument between the OUT/I and OUT/COM terminals. |

## Conversion characteristics

| 0mA to 20mA input or output | | 0V to 5V DC input | | -10V to 10V DC input or output | |
|---|---|---|---|---|---|
| **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** |
| 0 | 0.0mA | 0 | 0.0V | -2000 | -10.0V |
| 800 | 4.0mA | 400 | 0.5V | -1500 | -7.5V |
| 1600 | 8.0mA | 800 | 1.0V | -1000 | -5.0V |
| 2400 | 12.0mA | 1200 | 1.5V | -500 | -2.5V |
| 3200 | 16.0mA | 1600 | 2.0V | 0 | 0.0V |
| 4000 | 20.0mA | 2000 | 2.5V | +500 | +2.5V |
| | | 2400 | 3.0V | +1000 | +5.0V |
| | | 2800 | 3.5V | +1500 | +7.5V |
| | | 3200 | 4.0V | +2000 | +10.0V |
| | | 3600 | 4.5V | | |
| | | 4000 | 5.0V | | |

**Thermocouple input:**

| Above terminal temperature | | Below terminal temperature | |
|---|---|---|---|
| **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** |
| 25 | 25°C | 0 | 0°C |
| 250 | 250°C | -25 | -25°C |

| Above terminal temperature | | Below terminal temperature | |
|---|---|---|---|
| **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** |
| 350 | 350°C | -50 | -50°C |
| 500 | 500°C | -75 | -75°C |
| 750 | 750°C | -100 | -100°C |
| 1000 | 1000°C | | |

Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP0-A21 Analog I/O Unit Technical Manual”

## Example

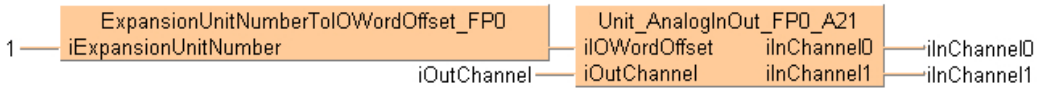
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | iOutChannel | INT | 0 |
| 2 | VAR | iIOWordOffset | INT | 0 |
| 3 | VAR | iInChannel0 | INT | 0 |
| 4 | VAR | iInChannel1 | INT | 0 |

### LD body

Use **ExpansionUnitNumberToIOWordOffset_FP0** or **ExpansionUnitNumberToIOWordOffset_FPX_FP0** to calculate the word offset of the analog unit connected to the CPU.



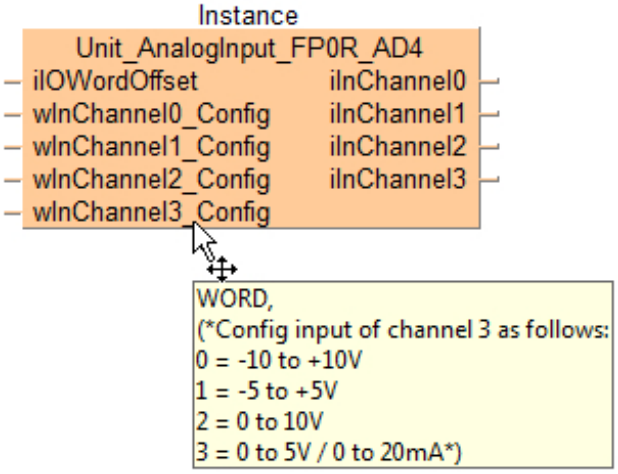
# Unit_AnalogInput_FP0R_AD4

Function block to read from an FP0R-AD4 unit.

This function block reads converted digital values from the analog input channels of the analog unit. The converted digital values are stored per channel in the output variables **iInChannel0** to **iInChannel3**.

The analog input ranges are also set with this function block.

The number of channels must be set with the DIP switches.




## Parameters

Input

**iIOWordOffset** (INT)

Set the offset of the first WX/WY address of the analog unit according to its installation position.
For analog expansion units connected directly to the CPU (without adapter): Use **ExpansionUnitToIOWordOffset_FP0** or make the following settings: 2 (WX2/WY2) for unit number 1, 4 (WX4/WY4) for unit number 2, 6 (WX6/WY6) for unit number 3
For analog expansion units connected to the CPU via an adapter: Use **ExpansionUnitToIOWordOffset_FPX_FP0** or select the offset from the table.

| **Unit position relative to the adapter** | **Adapter position relative to the CPU** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **1st unit** | **2nd unit** | **3rd unit** | **4th unit** | **5th unit** | **6th unit** | **7th unit** | **8th unit** |
| 1st unit | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 2nd unit | 32 | 42 | 52 | 62 | 72 | 82 | 92 | 102 |
| 3rd unit | 34 | 44 | 54 | 64 | 74 | 84 | 94 | 104 |

**wInChannel0_Config** to **wInChannel3_Config** (WORD)

Set the voltage or current range for the analog input channel.

Values:

0: -10 to +10V

1: -5 to +5V

2: 0 to 10V

3: 0 to 5V, 0 to 20mA (depending on wiring method)

Output

**iChannel0** to **iChannel3** (INT)

Returns the converted digital data from the analog unit by channel.

Values:

For -10 to +10V, -5 to +5V: -8000 to +8000

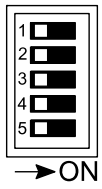
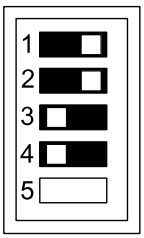
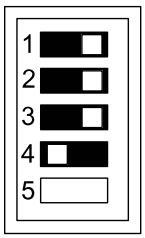
For 0 to 10V, 0 to 5 V, 0 to 20mA: 0 to 16000

## DIP switch settings

DIP switches 1 and 2 must be ON to use 14-bit mode. DIP switches 3 and 4 are used to set the number of channels, and DIP switch 5 is used to turn averaging on or off.

The DIP switch settings will become effective when the power is turned from OFF to ON.

**Number of channels:**

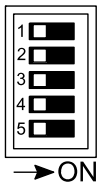
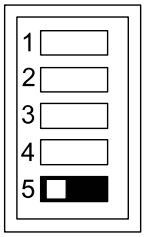
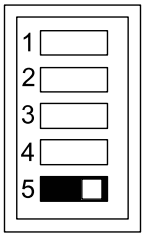
| | 2 (0 and 1) | 4 (0 to 3) |
|---|---|---|
|  |  |  |



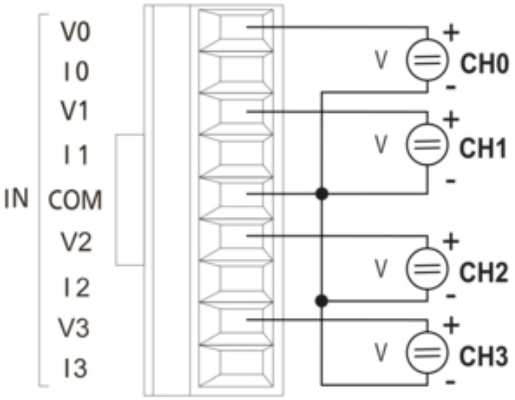
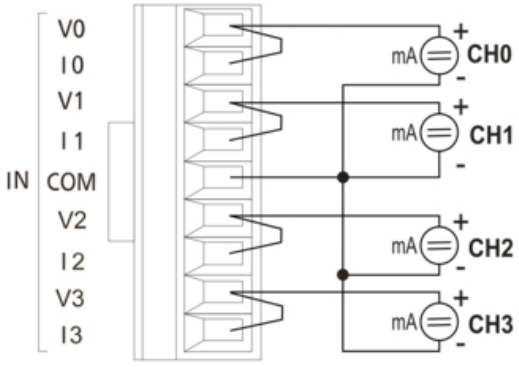
**Input averaging:**

No averaging: Conversion data is set for the specified input contact point area for each A/D conversion, on each channel.

Averaging: On each channel, for each A/D conversion, the maximum and minimum values from the data of the last ten times are excluded, the data from the other eight times is averaged and the result is set.

| | No averaging | Averaging |
|---|---|---|
|   |   |   |

## Input wiring

| Voltage input | Current input |
|---|---|
|   Connect the input instrument between the V and the COM terminal. |   Connect the V and the I terminals. Connect the input instrument between the bridge the and COM terminal. |

## Conversion characteristics

| -10V to +10V DC input | | -5V to +5V DC input | | 0V to 5V DC input | |
|---|---|---|---|---|---|
| Digital value (INT) | Analog value | Digital value (INT) | Analog value | Digital value (INT) | Analog value |
| -8000 | -10V | -8000 | -5V | 0 | 0.0V |
| -4000 | -5V | -4000 | -2.5V | 4000 | 1.25V |
| 0 | 0V | 0 | 0V | 8000 | 2.5V |
| +4000 | +5V | +4000 | +2.5V | 8000 | 3.75V |
| +8000 | +10V | +8000 | +5V | 16000 | 5.0V |

| 0V to 10V DC input | | 0mA to 20mA input | |
|---|---|---|---|
| Digital value (INT) | Analog value | Digital value (INT) | Analog value |
| 0 | 0.0V | 0 | 0.0mA |
| 4000 | 2.5V | 3200 | 4.0mA |
| 8000 | 5.0V | 6400 | 8.0mA |

| 0V to 10V DC input | | 0mA to 20mA input | |
|---|---|---|---|
| **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** |
| 12000 | 7.5V | 9600 | 12.0mA |
| 16000 | 10.0V | 12800 | 16.0mA |
| | | 16000 | 20.0mA |

Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP0R Analog I/O Unit User's Manual”

## Example

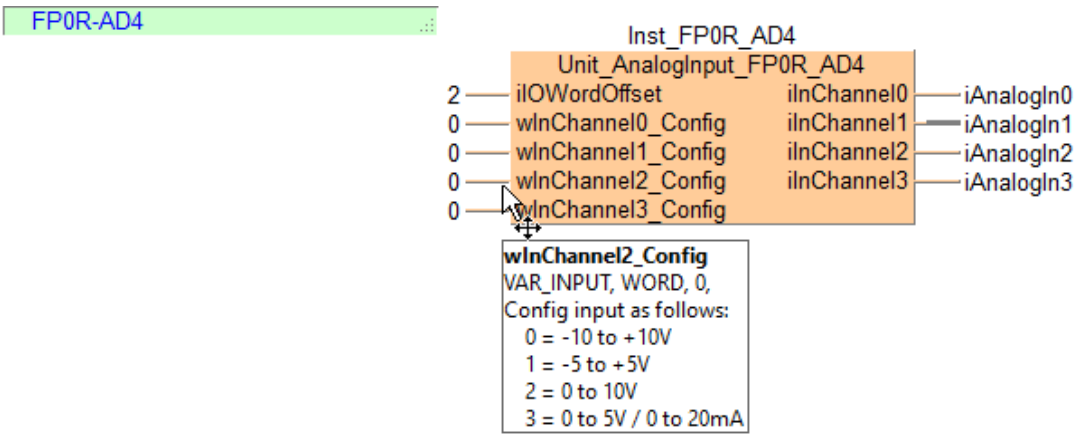
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | Inst_FP0R_AD4 | Unit_AnalogInput_FP0R_AD4 | |
| 2 | VAR | iAnalogIn0 | INT | 0 |
| 3 | VAR | iAnalogIn1 | INT | 0 |
| 4 | VAR | iAnalogIn2 | INT | 0 |
| 5 | VAR | iAnalogIn3 | INT | 0 |

### LD body

Use **ExpansionUnitNumberToIOWordOffset_FP0** or **ExpansionUnitNumberToIOWordOffset_FPX_FP0** to calculate the word offset of the analog unit connected to the CPU.



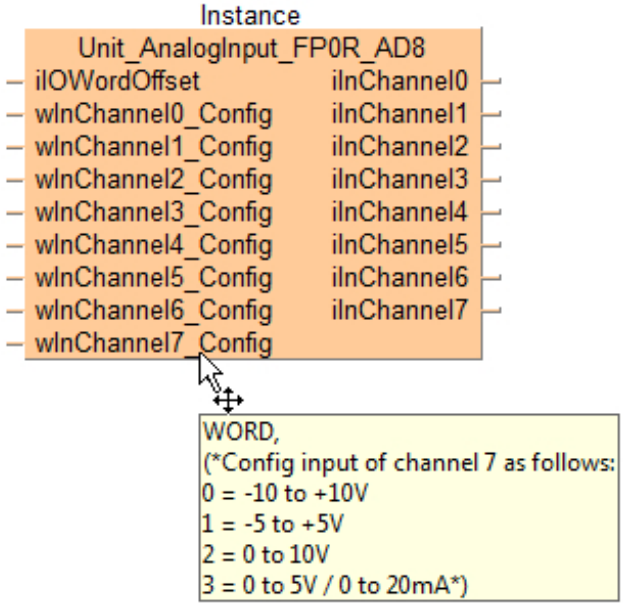
# Unit_AnalogInput_FP0R_AD8

Function block to read from an FP0R-AD8 unit.

This function block reads converted digital values from the analog input channels of the analog unit. The converted digital values from the analog unit are stored per channel in the output variables **iInChannel0** to **iInChannel7**.

The analog input ranges are also set with this function block.

The number of channels must be set with the DIP switches.




## Parameters

Input

**iIOWordOffset** (INT)

Set the offset of the first WX/WY address of the analog unit according to its installation position.

For analog expansion units connected directly to the CPU (without adapter): Use **ExpansionUnitToIOWordOffset_FP0** or make the following settings: 2 (WX2/WY2) for unit number 1, 4 (WX4/WY4) for unit number 2, 6 (WX6/WY6) for unit number 3

For analog expansion units connected to the CPU via an adapter: Use **ExpansionUnitToIOWordOffset_FPX_FP0** or select the offset from the table.

| Unit position relative to the adapter | Adapter position relative to the CPU | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1st unit | 2nd unit | 3rd unit | 4th unit | 5th unit | 6th unit | 7th unit | 8th unit |
| 1st unit | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 2nd unit | 32 | 42 | 52 | 62 | 72 | 82 | 92 | 102 |
| 3rd unit | 34 | 44 | 54 | 64 | 74 | 84 | 94 | 104 |

**wInChannel0_Config** to **wInChannel7_Config** (WORD)

Set the voltage or current range for the analog input channel.

Values:

0: -10 to +10V

1: -5 to +5V

2: 0 to 10V

3: 0 to 5V, 0 to 20mA (depending on wiring method)

Output

**iChannel0** to **iChannel7** (INT)

Returns the converted digital data from the analog unit by channel.

Values:

For -10 to +10V, -5 to +5V: -8000 to +8000

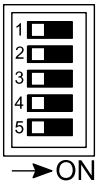
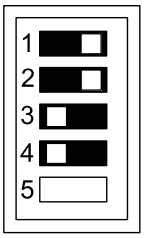
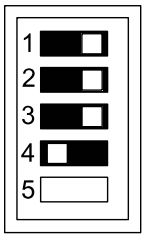
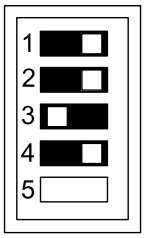
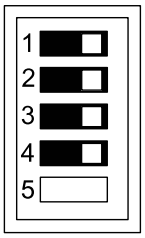
For 0 to 10V, 0 to 5 V, 0 to 20mA: 0 to 16000

## DIP switch settings

DIP switches 1 and 2 must be ON to use 14-bit mode. DIP switches 3 and 4 are used to set the number of channels, and DIP switch 5 is used to turn averaging on or off.

The DIP switch settings will become effective when the power is turned from OFF to ON.

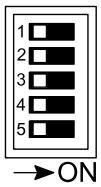
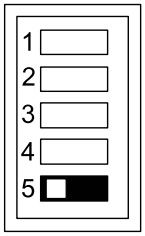
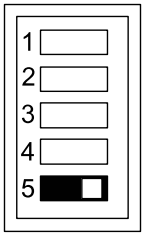
**Number of channels:**

| | 2 (0 and 1) | 4 (0 to 3) | 6 (0 to 5) | 8 (0 to 7) |
|---|---|---|---|---|
|  |  |  |  |  |

**Input averaging:**

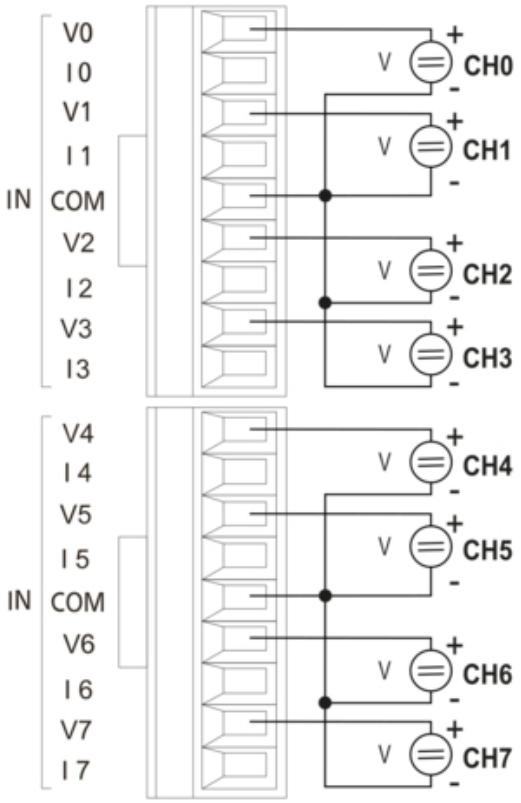
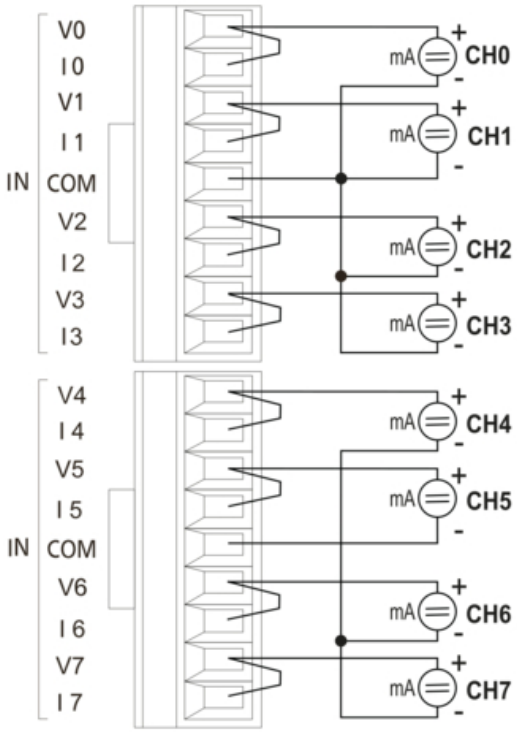
No averaging: Conversion data is set for the specified input contact point area for each A/D conversion, on each channel.

Averaging: On each channel, for each A/D conversion, the maximum and minimum values from the data of the last ten times are excluded, the data from the other eight times is averaged and the result is set.

| | No averaging | Averaging |
|---|---|---|
|  |  |  |



## Input wiring

| Voltage input | Current input |
|---|---|
|  |  |
| Connect the input instrument between the V and the COM terminal. | Connect the V and the I terminals. Connect the input instrument between the bridge the and COM terminal. |

### Conversion characteristics

| -10V to +10V DC input | | -5V to +5V DC input | | 0V to 5V DC input | |
|---|---|---|---|---|---|
| Digital value (INT) | Analog value | Digital value (INT) | Analog value | Digital value (INT) | Analog value |
| -8000 | -10V | -8000 | -5V | 0 | 0.0V |
| -4000 | -5V | -4000 | -2.5V | 4000 | 1.25V |
| 0 | 0V | 0 | 0V | 8000 | 2.5V |
| +4000 | +5V | +4000 | +2.5V | 8000 | 3.75V |
| +8000 | +10V | +8000 | +5V | 16000 | 5.0V |

| 0V to 10V DC input | | 0mA to 20mA input | |
|---|---|---|---|
| Digital value (INT) | Analog value | Digital value (INT) | Analog value |
| 0 | 0.0V | 0 | 0.0mA |
| 4000 | 2.5V | 3200 | 4.0mA |
| 8000 | 5.0V | 6400 | 8.0mA |
| 12000 | 7.5V | 9600 | 12.0mA |
| 16000 | 10.0V | 12800 | 16.0mA |
| | | 16000 | 20.0mA |

Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP0R Analog I/O Unit User's Manual”

## Example

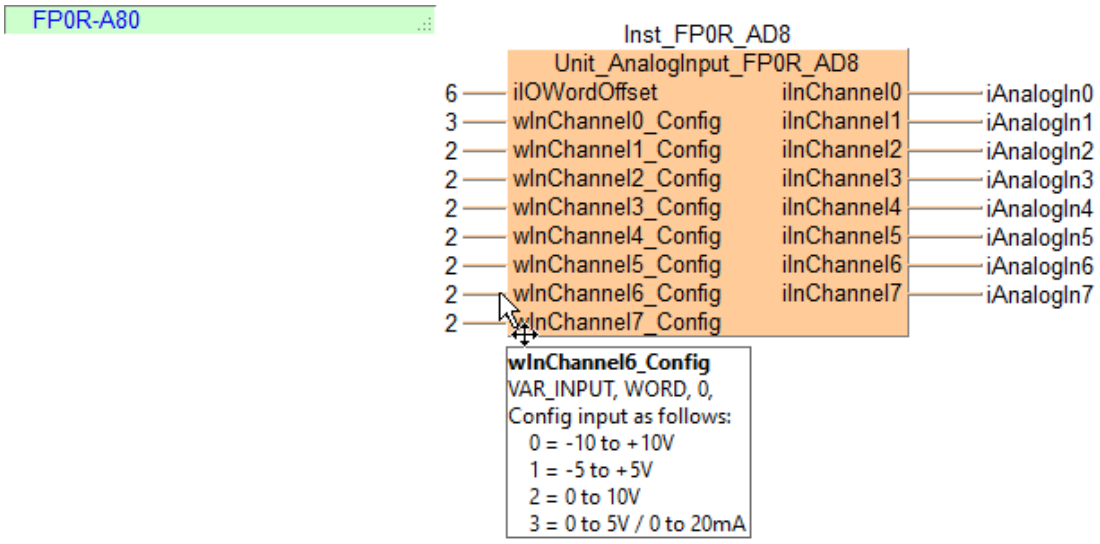
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | Inst_FP0R_AD8 | Unit_AnalogInput_FP0R_AD8 | |
| 2 | VAR | iAnalogIn0 | INT | 0 |
| 3 | VAR | iAnalogIn1 | INT | 0 |
| 4 | VAR | iAnalogIn2 | INT | 0 |
| 5 | VAR | iAnalogIn3 | INT | 0 |
| 6 | VAR | iAnalogIn4 | INT | 0 |
| 7 | VAR | iAnalogIn5 | INT | 0 |
| 8 | VAR | iAnalogIn6 | INT | 0 |
| 9 | VAR | iAnalogIn7 | INT | 0 |

## LD body

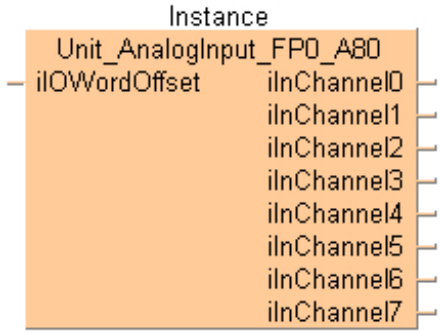
Use **ExpansionUnitNumberToIOWordOffset_FP0** or **ExpansionUnitNumberToIOWordOffset_FPX_FP0** to calculate the word offset of the analog unit connected to the CPU.



# Unit_AnalogInput_FP0_A80

Function block to read from an FP0-A80 unit.

This function block reads converted digital values from the analog input channels of the analog unit. The converted digital values are stored per channel in the output variables **iInChannel0** to **iInChannel7**.




## Parameters

Input

**iIOWordOffset** (INT)

Set the offset of the first WX/WY address of the analog unit according to its installation position.

For analog expansion units connected directly to the CPU (without adapter): Use **ExpansionUnitToIOWordOffset_FP0** or make the following settings: 2 (WX2/WY2) for unit number 1, 4 (WX4/WY4) for unit number 2, 6 (WX6/WY6) for unit number 3

For analog expansion units connected to the CPU via an adapter: Use **ExpansionUnitToIOWordOffset_FPX_FP0** or select the offset from the table.

| Unit position relative to the adapter | Adapter position relative to the CPU | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1st unit | 2nd unit | 3rd unit | 4th unit | 5th unit | 6th unit | 7th unit | 8th unit |
| 1st unit | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 2nd unit | 32 | 42 | 52 | 62 | 72 | 82 | 92 | 102 |
| 3rd unit | 34 | 44 | 54 | 64 | 74 | 84 | 94 | 104 |

Output

**iInChannel0** to **iInChannel7** (INT)

Returns the converted digital data from the analog unit by channel.

Values:

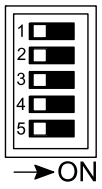
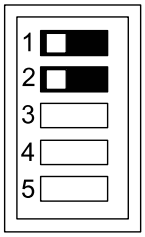
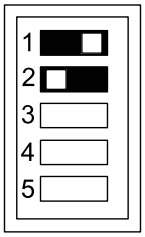
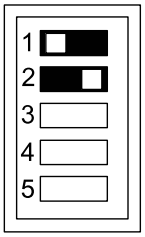
For 0 to 5V, 0 to 20mA : 0 to 4000 (depending on wiring method)

For -10 to +10V, -100 to +100mV: -2000 to +2000

## DIP switch settings

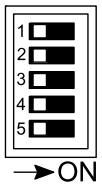
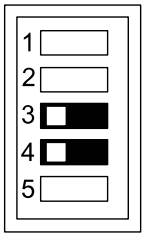
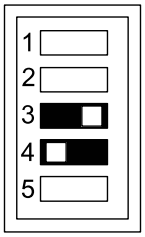
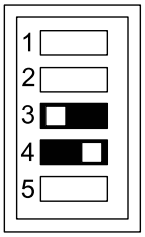
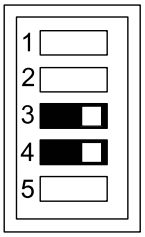
The DIP switch settings will become effective when the power is turned from OFF to ON.

**Input range:**

| | 0 to 5V, 0 to 20mA | -10 to +10V | -100 to +100mV |
|---|---|---|---|
|  |  |  |  |



**Number of channels:**

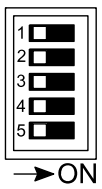
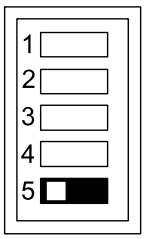
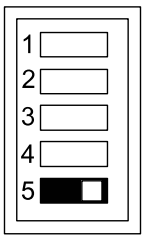
| | 2 (0 and 1) | 4 (0 to 3) | 6 (0 to 5) | 8 (0 to 7) |
|---|---|---|---|---|
|  |  |  |  |  |



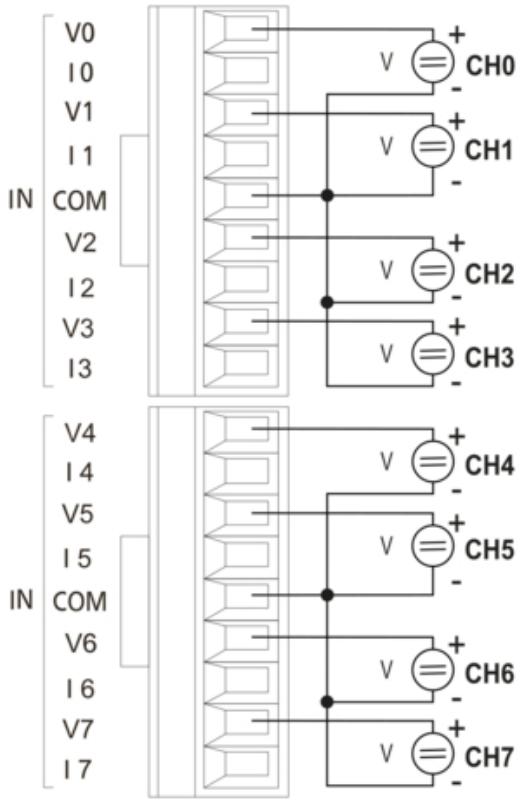
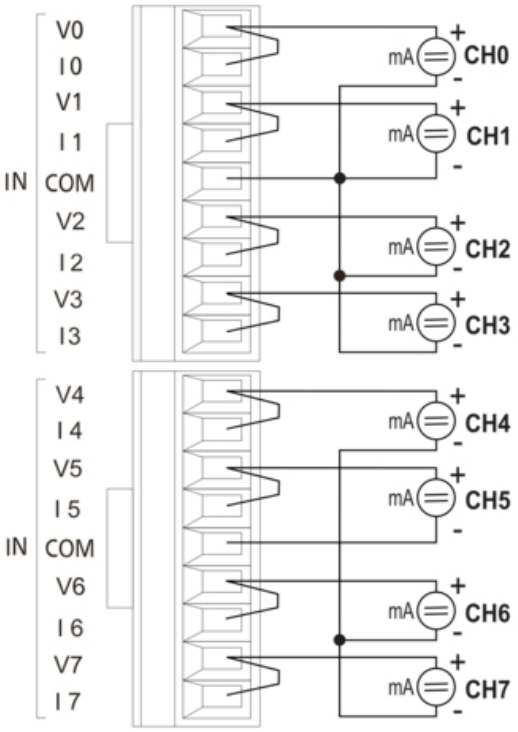
**Input averaging:**

No averaging: Conversion data is set for the specified input contact point area for each A/D conversion, on each channel.

Averaging: On each channel, for each A/D conversion, the maximum and minimum values from the data of the last ten times are excluded, the data from the other eight times is averaged and the result is set.

| | No averaging | Averaging |
|---|---|---|
|  |  |  |

## Input wiring

| Voltage input | Current input |
|---|---|
|  |  |
| Connect the input instrument between the V and the COM terminals. | Connect the V and the I terminals. Connect the input instrument between the bridge and the COM terminal. |

## Conversion characteristics

| -10V to +10V DC input | | 0V to 5V DC input | | -100mV to +100mV DC input | | 0mA to 20mA input | |
|---|---|---|---|---|---|---|---|
| **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** |
| -2000 | -10.0V | 0 | 0.0V | -2000 | -100.0mV | 0 | 0.0mA |
| -1500 | -7.5V | 400 | 0.5V | -1500 | -75.0mV | 800 | 4.0mA |
| -1000 | -5.0V | 800 | 1.0V | -1000 | -50.0mV | 1600 | 8.0mA |
| -500 | -2.5V | 1200 | 1.5V | -500 | -25.0mV | 2400 | 12.0mA |
| 0 | 0.0V | 1600 | 2.0V | 0 | 0.0mV | 3200 | 16.0mA |
| +500 | +2.5V | 2000 | 2.5V | +500 | +25.0mV | 4000 | 20.0mA |
| +1000 | +5.0V | 2400 | 3.0V | +1000 | +50.0mV | | |
| +1500 | +7.5V | 2800 | 3.5V | +1500 | +75.0mV | | |
| +2000 | +10.0V | 3200 | 4.0V | +2000 | +100.0mV | | |

| -10V to +10V DC input | | 0V to 5V DC input | | -100mV to +100mV DC input | | 0mA to 20mA input | |
|---|---|---|---|---|---|---|---|
| **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** |
| | | 3600 | 4.5V | | | | |
| | | 40000 | 5.0V | | | | |

Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP0 A/D Converter Unit Technical Manual”

## Example

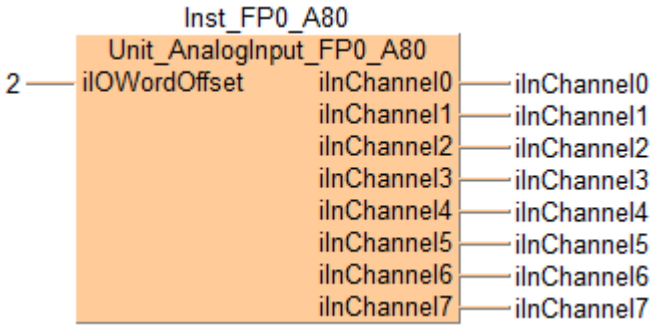
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | Inst_FP0_A80 | Unit_AnalogInput_FP0_A80 | |
| 2 | VAR | iInChannel0 | INT | 0 |
| 3 | VAR | iInChannel1 | INT | 0 |
| 4 | VAR | iInChannel2 | INT | 0 |
| 5 | VAR | iInChannel3 | INT | 0 |
| 6 | VAR | iInChannel4 | INT | 0 |
| 7 | VAR | iInChannel5 | INT | 0 |
| 8 | VAR | iInChannel6 | INT | 0 |
| 9 | VAR | iInChannel7 | INT | 0 |

### LD body

Use **ExpansionUnitNumberToIOWordOffset_FP0** or **ExpansionUnitNumberToIOWordOffset_FPX_FP0** to calculate the word offset of the analog unit connected to the CPU.


Inst_FP0_A80
Unit_AnalogInput_FP0_A80
2
iIOWordOffset
iInChannel0
iInChannel1
iInChannel2
iInChannel3
iInChannel4
iInChannel5
iInChannel6
iInChannel7
iInChannel0
iInChannel1
iInChannel2
iInChannel3
iInChannel4
iInChannel5
iInChannel6
iInChannel7

# Unit_AnalogInput_FP0_RTD_INT

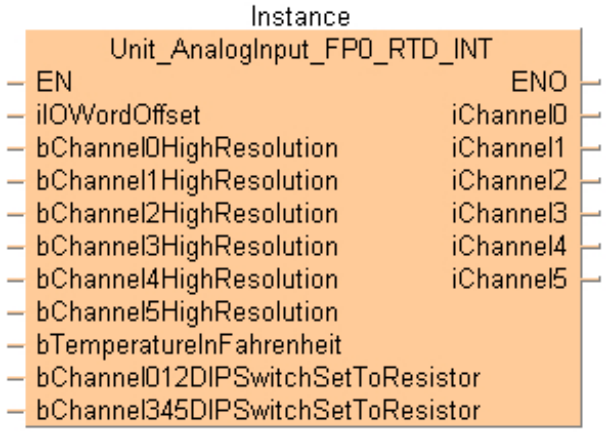
Function block to read from an FP0-RTD6 unit (converted digital values are of data type INT).

This function block reads converted digital values from the analog input channels of the analog unit. The converted digital values from the analog unit are stored per channel in the output variables **iChannel0** to **iChannel5**.

For measuring RTD input data, you can use the following devices: Pt100 (according to IEC751), Pt1000 (according to IEC751), Ni1000 (according to DIN43760), or a resistor.

The measurement device must be set with the DIP switches and in the function block.

The sampling cycle must be set with the DIP switches.




## Parameters

Input

**iIOWordOffset** (INT)

Set the offset of the first WX/WY address of the analog unit according to its installation position.
For analog expansion units connected directly to the CPU (without adapter): Use **ExpansionUnitToIOWordOffset_FP0** or make the following settings: 2 (WX2/WY2) for unit number 1, 4 (WX4/WY4) for unit number 2, 6 (WX6/WY6) for unit number 3
For analog expansion units connected to the CPU via an adapter: Use **ExpansionUnitToIOWordOffset_FPX_FP0** or select the offset from the table.

| Unit position relative to the adapter | Adapter position relative to the CPU | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1st unit | 2nd unit | 3rd unit | 4th unit | 5th unit | 6th unit | 7th unit | 8th unit |
| 1st unit | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 2nd unit | 32 | 42 | 52 | 62 | 72 | 82 | 92 | 102 |
| 3rd unit | 34 | 44 | 54 | 64 | 74 | 84 | 94 | 104 |

**bChannel0HighResolution** to **bChannel5HighResolution** (BOOL)

Set the resolution of the corresponding channel.
Values:
TRUE: High resolution (0.01K/0.01°F/0.1Ω)
FALSE: Low resolution (0.1K/0.1°F/1Ω)

Do not change this value during runtime. Otherwise conversion will be inaccurate for 1s.

**bTemperatureInFahrenheit** (BOOL)

Set the temperature unit.
Values:
TRUE: °F
FALSE: °C

**bChannel012DIPSwitchSetToResistor**, **bChannel345DIPSwitchSetToResistor** (BOOL)

Set the RTD device.
Values:
TRUE: Resistor
FALSE: Pt100, Pt1000, Ni1000

The setting must match the DIP switch settings.

Output

**iChannel0** to **iChannel5** (INT)

Returns the converted digital data from the analog unit by channel.
Example:

- Thermocouple, low resolution: 20.12°C → 201 (measurement value out of range: 8191)
- Thermocouple, high resolution 20.12°C → 2012 (measurement value out of range: 8191)
- Resistor, low resolution: 25Ω → 25 (measurement value out of range: 16383)
- Resistor, high resolution: 25.4Ω → 254 (measurement value out of range: 16383)

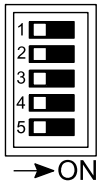
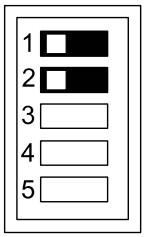
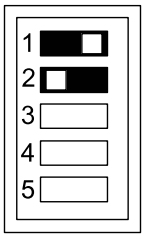
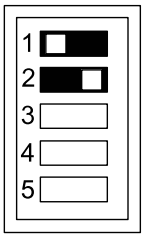
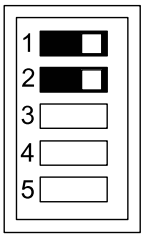
## Note

- Between power ON and the first valid conversion data, the digital value will be 8191 or 16383. When programming, be sure not to use the data obtained during this period.
- When the RTD is broken, the digital value will change to 8191 or 16383. When programming avoid any risks resulting from a broken RTD. A broken RTD needs to be replaced.

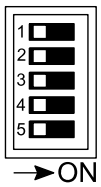
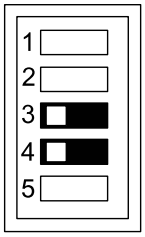
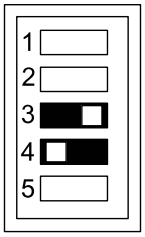
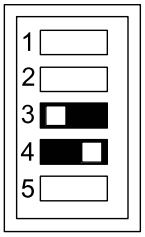
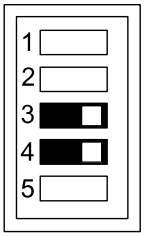
## DIP switch settings

The DIP switch settings will become effective when the power is turned from OFF to ON.

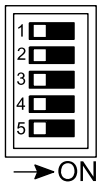
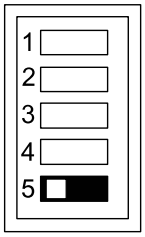
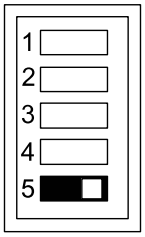
**Measurement device, channel 0 to 2**

| | Pt100 | Pt1000 | Ni1000 | Resistor |
|---|---|---|---|---|
|  |  |  |  |  |

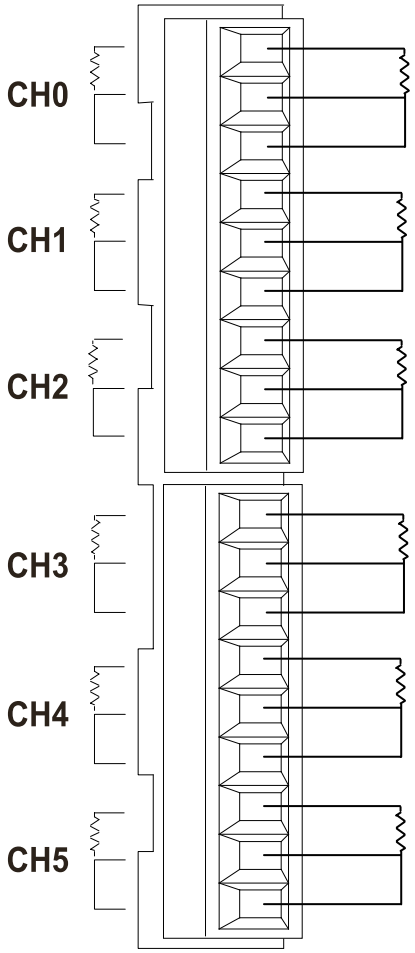
**Measurement device, channel 3 to 5**

| | Pt100 | Pt1000 | Ni1000 | Resistor |
|---|---|---|---|---|
|  |  |  |  |  |

**Sampling cycle**

| | 0.1s | 1s |
|---|---|---|
|  |  |  |

### Input wiring




### Conversion characteristics

**Resolution: 0.1K/0.1F, 1Ω**

| Pt100 | | Pt1000 | |
|---|---|---|---|
| °C/°*F* | | °C/°*F* | |
| **Analog value** | **Digital value** | **Analog value** | **Digital value** |
| -200.0/*-328.0* | -2000/*-3280* | -200.0/*-328.0* | -2000/*-3280* |
| +500.0/*+800.0* | +5000/*+8000* | +300.0/*+572.0* | +3000/*+5720* |

| Ni1000 | | Resistor | |
|---|---|---|---|
| °C/°*F* | | Ω | |
| **Analog value** | **Digital value** | **Analog value** | **Digital value** |
| -30.0/*-22.0* | -300/*-220* | +20 | +20 |
| +150.0/+302.0 | +1500/+3020 | +2200 | +2200 |

**Resolution: 0.01K/0.01F, 0.1Ω**

| Pt100 | | Pt1000 | |
|---|---|---|---|
| °C/°F | | °C/°F | |
| **Analog value** | **Digital value** | **Analog value** | **Digital value** |
| -80.00/*-80.00* | -8000/*-8000* | -80.00/*-80.00* | -8000/*-8000* |
| +80.00/*+80.00* | +8000/*+8000* | +80.00/*+80.00* | +8000/*+8000* |

| Ni1000 | | Resistor | |
|---|---|---|---|
| °C/°F | | Ω | |
| **Analog value** | **Digital value** | **Analog value** | **Digital value** |
| -30.00/*-22.00* | -3000/*-2200* | +20.0 | +200 |
| +80.00/*+80.00* | +8000/*+8000* | +1630.0 | +16300 |

Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP0 RTD Unit Technical Manual”

## Example

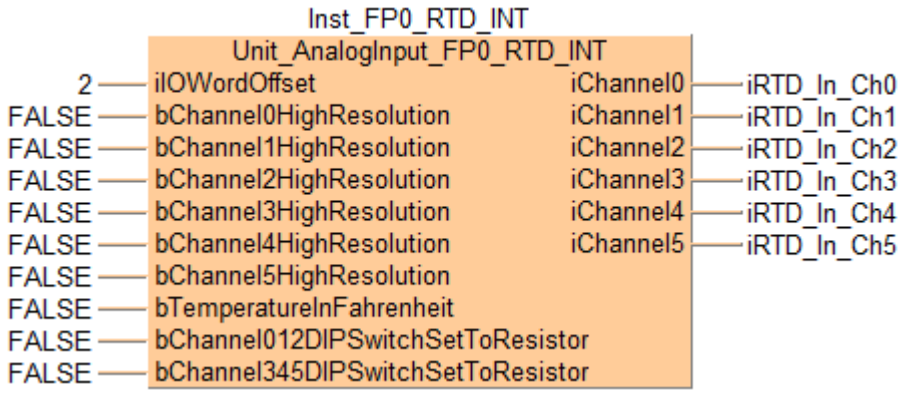
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | Inst_FP0_RTD_INT | Unit_AnalogInput_FP0_RTD_INT | |
| 2 | VAR | iRTD_In_Ch0 | INT | 0 |
| 3 | VAR | iRTD_In_Ch1 | INT | 0 |
| 4 | VAR | iRTD_In_Ch2 | INT | 0 |
| 5 | VAR | iRTD_In_Ch3 | INT | 0 |
| 6 | VAR | iRTD_In_Ch4 | INT | 0 |
| 7 | VAR | iRTD_In_Ch5 | INT | 0 |

## LD body

Use **ExpansionUnitNumberToIOWordOffset_FP0** or **ExpansionUnitNumberToIOWordOffset_FPX_FP0** to calculate the word offset of the analog unit connected to the CPU.



# Unit_AnalogInput_FP0_RTD_REAL

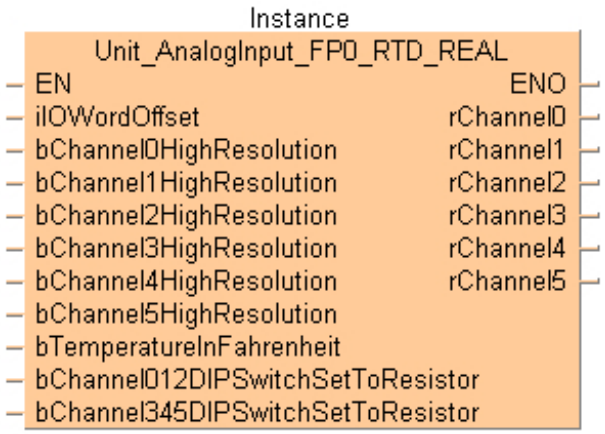
Function block to read from an FP0-RTD6 unit (converted digital values are of data type REAL)

This function block reads converted digital values from the analog input channels of the analog unit. The converted digital values from the analog unit are stored per channel in the output variables **rChannel0** to **rChannel5**.

For measuring RTD input data, you can use the following devices: Pt100 (according to IEC751), Pt1000 (according to IEC751), Ni1000 (according to DIN43760), or a resistor.

The measurement device must be set with the DIP switches and in the function block.

The sampling cycle must be set with the DIP switches.




## Parameters

Input

**iIOWordOffset** (INT)

Set the offset of the first WX/WY address of the analog unit according to its installation position.
For analog expansion units connected directly to the CPU (without adapter): Use **ExpansionUnitToIOWordOffset_FP0** or make the following settings: 2 (WX2/WY2) for unit number 1, 4 (WX4/WY4) for unit number 2, 6 (WX6/WY6) for unit number 3
For analog expansion units connected to the CPU via an adapter: Use **ExpansionUnitToIOWordOffset_FPX_FP0** or select the offset from the table.

| **Unit position relative to the adapter** | **Adapter position relative to the CPU** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **1st unit** | **2nd unit** | **3rd unit** | **4th unit** | **5th unit** | **6th unit** | **7th unit** | **8th unit** |
| 1st unit | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 2nd unit | 32 | 42 | 52 | 62 | 72 | 82 | 92 | 102 |
| 3rd unit | 34 | 44 | 54 | 64 | 74 | 84 | 94 | 104 |

**bChannel0HighResolution** to **bChannel5HighResolution** (BOOL)

Set the resolution of the corresponding channel.
Values:
TRUE: High resolution (0.01K/0.01°F/0.1Ω)
FALSE: Low resolution (0.1K/0.1°F/1Ω)

Do not change this value during runtime. Otherwise conversion will be inaccurate for 1s.

**bTemperatureInFahrenheit** (BOOL)

Set the temperature unit.
Values:
TRUE: °F
FALSE: °C

**bChannel012DIPSwitchSetToResistor**, **bChannel345DIPSwitchSetToResistor** (BOOL)

Set the RTD device.
Values:
TRUE: Resistor
FALSE: Pt100, Pt1000, Ni1000

The setting must match the DIP switch settings.

Output

**rChannel0** to **rChannel5** (REAL)

Returns the converted digital data from the analog unit by channel.
Example:

- Thermocouple, low resolution: 20.12°C → 20.1 (measurement value out of range: 819.1)
- Thermocouple, high resolution 20.12°C → 20.12 (measurement value out of range: 81.91)
- Resistor, low resolution: 25Ω → 25 (measurement value out of range: 16383)
- Resistor, high resolution: 25.4Ω → 25.4 (measurement value out of range: 1638.3)

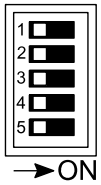
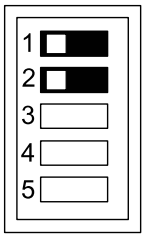
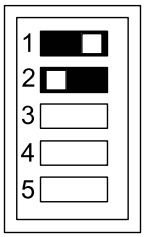
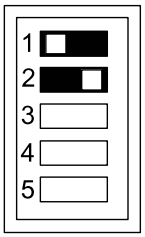
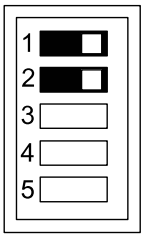
## Note

- Between power ON and the first valid conversion data, the digital value will be 8191 or 16383. When programming, be sure not to use the data obtained during this period.
- When the RTD is broken, the digital value will change to 8191 or 16383. When programming avoid any risks resulting from a broken RTD. A broken RTD needs to be replaced.

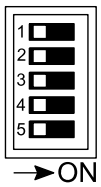
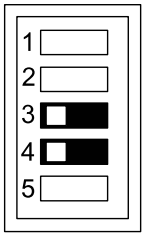
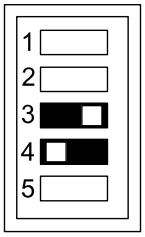
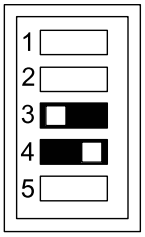
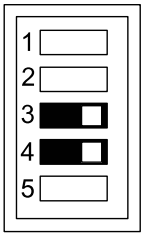
## DIP switch settings

The DIP switch settings will become effective when the power is turned from OFF to ON.

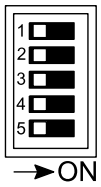
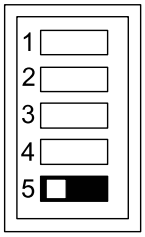
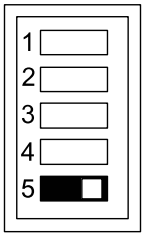
**Measurement device, channel 0 to 2**

| | Pt100 | Pt1000 | Ni1000 | Resistor |
|---|---|---|---|---|
|  |  |  |  |  |

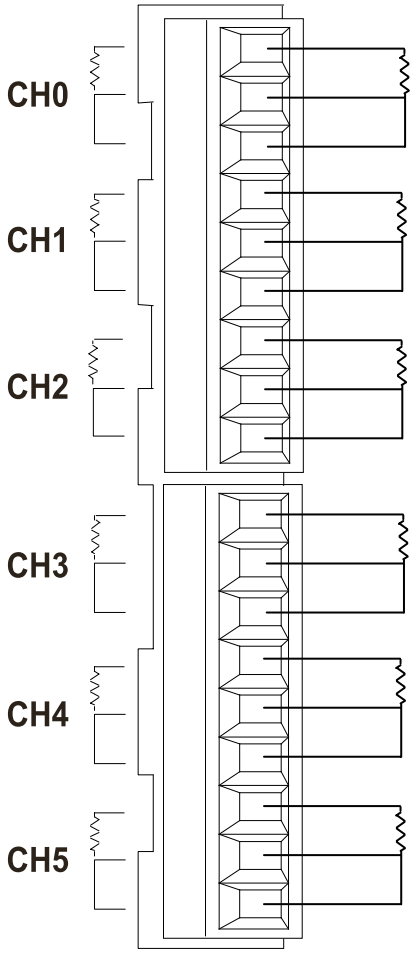
**Measurement device, channel 3 to 5**

| | Pt100 | Pt1000 | Ni1000 | Resistor |
|---|---|---|---|---|
|  |  |  |  |  |

**Sampling cycle**

| | 0.1s | 1s |
|---|---|---|
|  |  |  |

### Input wiring




### Conversion characteristics

**Resolution: 0.1K/0.1F, 1Ω**

| Pt100 | | Pt1000 | |
|---|---|---|---|
| °C/*°F* | | °C/*°F* | |
| **Analog value** | **Digital value** | **Analog value** | **Digital value** |
| -200.0/*-328.0* | -2000/*-3280* | -200.0/*-328.0* | -2000/*-3280* |
| +500.0/*+800.0* | +5000/*+8000* | +300.0/*+572.0* | +3000/*+5720* |

| Ni1000 | | Resistor | |
|---|---|---|---|
| °C/*°F* | | Ω | |
| **Analog value** | **Digital value** | **Analog value** | **Digital value** |
| -30.0/*-22.0* | -300/*-220* | +20 | +20 |
| +150.0/+302.0 | +1500/+3020 | +2200 | +2200 |

**Resolution: 0.01K/0.01F, 0.1Ω**

| Pt100 | | Pt1000 | |
|---|---|---|---|
| °C/°F | | °C/°F | |
| **Analog value** | **Digital value** | **Analog value** | **Digital value** |
| -80.00/*-80.00* | -8000/*-8000* | -80.00/*-80.00* | -8000/*-8000* |
| +80.00/*+80.00* | +8000/*+8000* | +80.00/*+80.00* | +8000/*+8000* |

| Ni1000 | | Resistor | |
|---|---|---|---|
| °C/°F | | Ω | |
| **Analog value** | **Digital value** | **Analog value** | **Digital value** |
| -30.00/*-22.00* | -3000/*-2200* | +20.0 | +200 |
| +80.00/*+80.00* | +8000/*+8000* | +1630.0 | +16300 |

Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP0 RTD Unit Technical Manual”

## Example

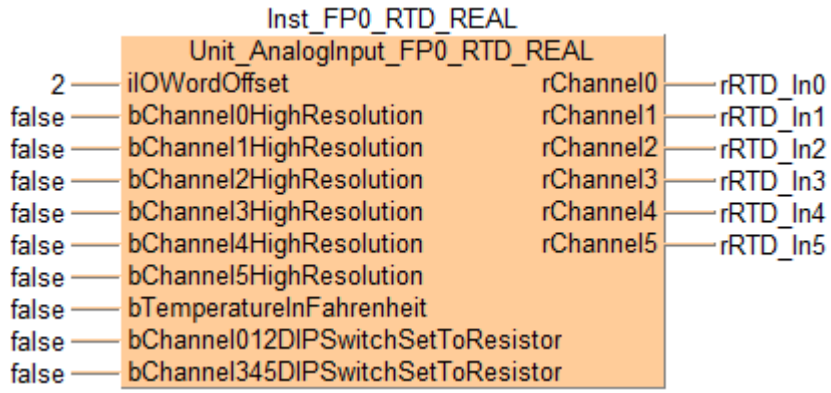
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | Inst_FP0_RTD_REAL | Unit_AnalogInput_FP0_RTD_REAL | |
| 2 | VAR | rRTD_In0 | REAL | 0 |
| 3 | VAR | rRTD_In1 | REAL | 0 |
| 4 | VAR | rRTD_In2 | REAL | 0 |
| 5 | VAR | rRTD_In3 | REAL | 0 |
| 6 | VAR | rRTD_In4 | REAL | 0 |
| 7 | VAR | rRTD_In5 | REAL | 0 |

## LD body

Use **ExpansionUnitNumberToIOWordOffset_FP0** or **ExpansionUnitNumberToIOWordOffset_FPX_FP0** to calculate the word offset of the analog unit connected to the CPU.

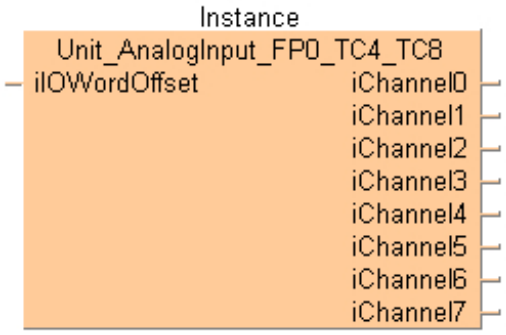


# Unit_AnalogInput_FP0_TC4_TC8

Function block to read from an FP0-TC4 or FP0-TC8 unit.

This function block reads converted digital values from the analog input channels of the analog unit. The converted digital values are stored per channel in the output variables **iInChannel0** to **iInChannel3** (FP0-TC4) or **iChannel0** to **iChannel7** (FP0-TC8).

The thermocouple type, the temperature unit (°C, °F), and the number of input channels must be set with the DIP switches.




## Parameters

Input

**iIOWordOffset** (INT)

Set the offset of the first WX/WY address of the analog unit according to its installation position.

For analog expansion units connected directly to the CPU (without adapter): Use **ExpansionUnitToIOWordOffset_FP0** or make the following settings: 2 (WX2/WY2) for unit number 1, 4 (WX4/WY4) for unit number 2, 6 (WX6/WY6) for unit number 3

For analog expansion units connected to the CPU via an adapter: Use **ExpansionUnitToIOWordOffset_FPX_FP0** or select the offset from the table.

| Unit position relative to the adapter | Adapter position relative to the CPU | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1st unit | 2nd unit | 3rd unit | 4th unit | 5th unit | 6th unit | 7th unit | 8th unit |
| 1st unit | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 2nd unit | 32 | 42 | 52 | 62 | 72 | 82 | 92 | 102 |
| 3rd unit | 34 | 44 | 54 | 64 | 74 | 84 | 94 | 104 |

Output

**iChannel0** to **iChannel7** (INT)

Returns the converted digital data from the analog unit by channel.

Values:

K, J type:

For -100.1°C to +500.1°C: -1001 to +5001

For -148.1°F to +790.1°F: -1481 to +7901

T type:

For -100.1°C to +400.1°C: -1001 to +4001

For -148.1°F to +752.1°F: -1481 to +7521

R type:

For 0°C to 1500.1°C: 0 to 15001

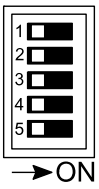
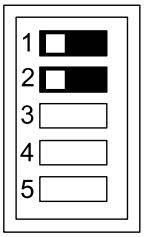
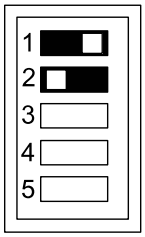
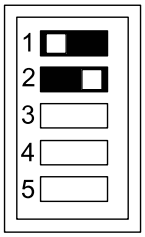
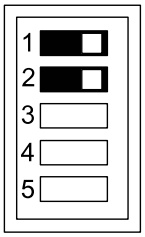
For 32°F to 1590.1°F: 320 to 15901

Wire broken: 8000 or 16000

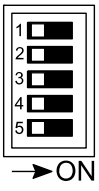
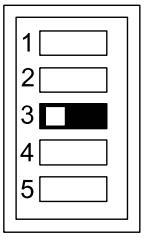
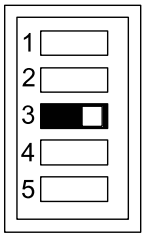
## DIP switch settings

The DIP switch settings will become effective when the power is turned from OFF to ON.

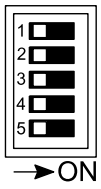
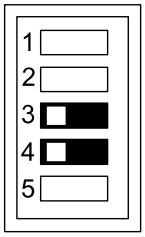
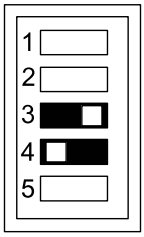
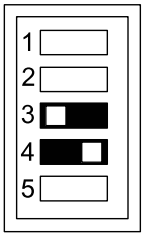
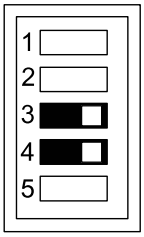
**Thermocouple type**

| | K | J | T | R |
|---|---|---|---|---|
|  |  |  |  |  |

**Temperature unit**

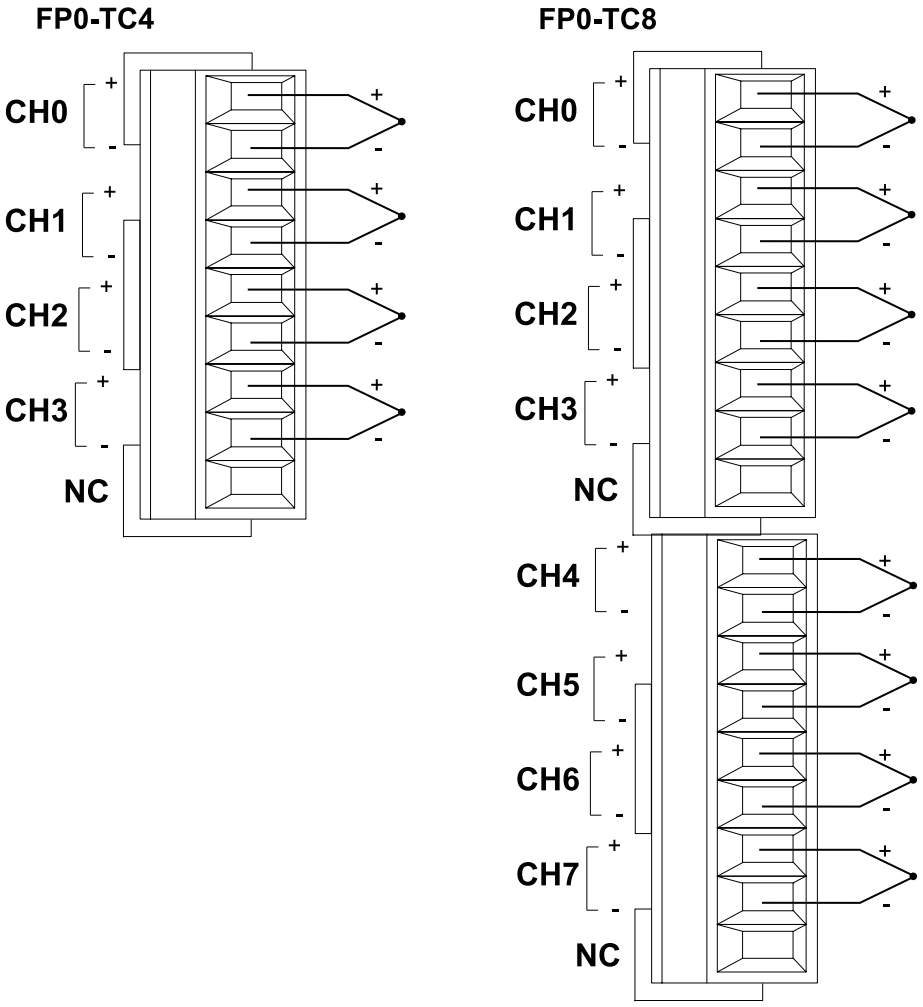
| | °C | °F |
|---|---|---|
|  |  |  |

**Number of channels**

| | 2 (0 and 1) | 4 (0 to 3) | 6 (0 to 5) | 8 (0 to 7) |
|---|---|---|---|---|
|  |  |  |  |  |



## Input wiring





Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP0 Thermocouple Unit Technical Manual”

## Example

### POU header

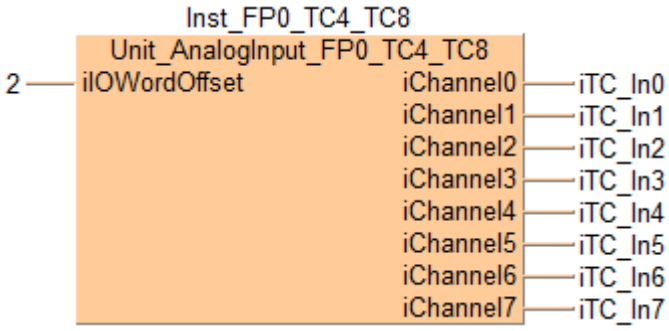
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | Inst_FP0_TC4_TC8 | Unit_AnalogInput_FP0_TC4_TC8 | |
| 2 | VAR | iTC_In0 | INT | 0 |
| 3 | VAR | iTC_In1 | INT | 0 |
| 4 | VAR | iTC_In2 | INT | 0 |
| 5 | VAR | iTC_In3 | INT | 0 |
| 6 | VAR | iTC_In4 | INT | 0 |
| 7 | VAR | iTC_In5 | INT | 0 |
| 8 | VAR | iTC_In6 | INT | 0 |
| 9 | VAR | iTC_In7 | INT | 0 |

### POU body

Use **ExpansionUnitNumberToIOWordOffset_FP0** or **ExpansionUnitNumberToIOWordOffset_FPX_FP0** to calculate the word offset of the analog unit connected to the CPU.

### LD body



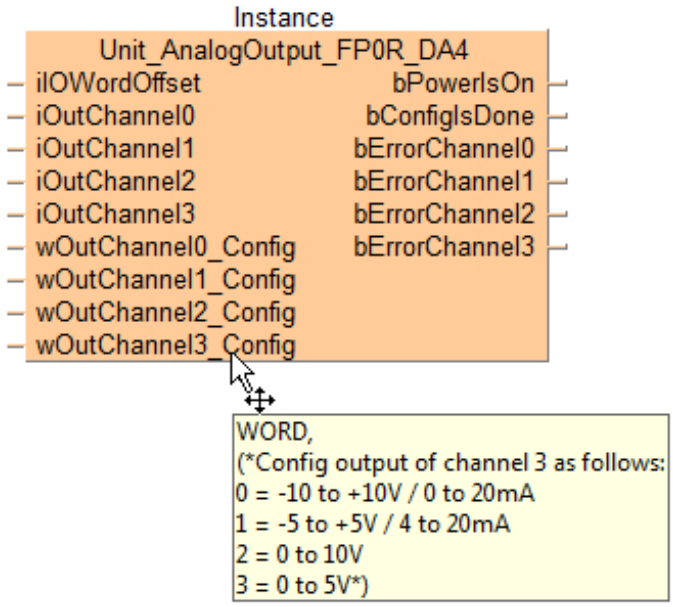
# Unit_AnalogOutput_FP0R_DA4

Function block to write to an FP0R-DA4 unit.

This function block writes digital data to the analog output channels of the analog unit. The digital values to be converted and output as analog values are specified at **iOutChannel0** to **iOutChannel3**.

The analog output ranges are also set with this function block.

The voltage or current output must be set with the DIP switches.




## Parameters

Input

**iIOWordOffset** (INT)

Set the offset of the first WX/WY address of the analog unit according to its installation position.
For analog expansion units connected directly to the CPU (without adapter): Use **ExpansionUnitToIOWordOffset_FP0** or make the following settings: 2 (WX2/WY2) for unit number 1, 4 (WX4/WY4) for unit number 2, 6 (WX6/WY6) for unit number 3
For analog expansion units connected to the CPU via an adapter: Use **ExpansionUnitToIOWordOffset_FPX_FP0** or select the offset from the table.

| Unit position relative to the adapter | Adapter position relative to the CPU | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1st unit | 2nd unit | 3rd unit | 4th unit | 5th unit | 6th unit | 7th unit | 8th unit |
| 1st unit | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 2nd unit | 32 | 42 | 52 | 62 | 72 | 82 | 92 | 102 |
| 3rd unit | 34 | 44 | 54 | 64 | 74 | 84 | 94 | 104 |

**iOutChannel0** to **iOutChannel3** (INT)

Set the digital value to be converted and output by the analog unit.
Values:
For -10 to +10V, -5 to +5V: -8000 to +8000
For 0 to 10V, 0 to 5 V, 0 to 20mA, 4 to 20mA: 0 to 16000

**wOutChannel0_Config** to **wOutChannel3_Config** (WORD)

Set the voltage or current range for the analog output channel.
Values:
0: -10 to +10V, 0 to 20mA (depending on wiring method)
1: -5 to +5V, 4 to 20mA (depending on wiring method)
2: 0 to 10V
3: 0 to 5V

Output

**bPowerIsOn** (BOOL)

Unit status: TRUE when the power is on.

**bConfigIsDone** (BOOL)

TRUE when I/O configuration is completed and the unit is ready.

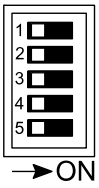
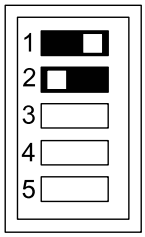
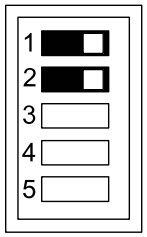
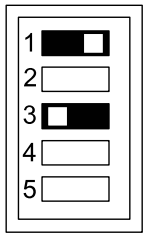
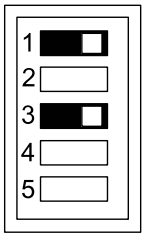
**bErrorChannel0** to **bErrorChannel3** (BOOL)

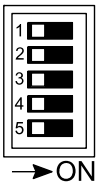
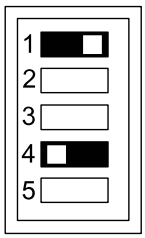
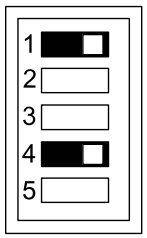
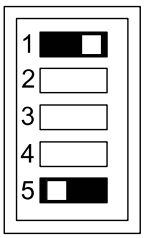
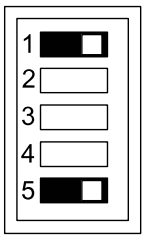
Channel status: TRUE if there is an error.

## DIP switch settings

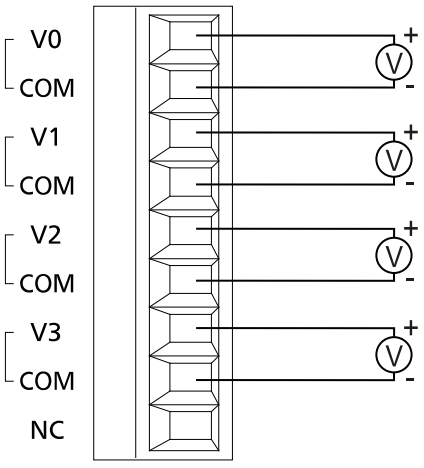
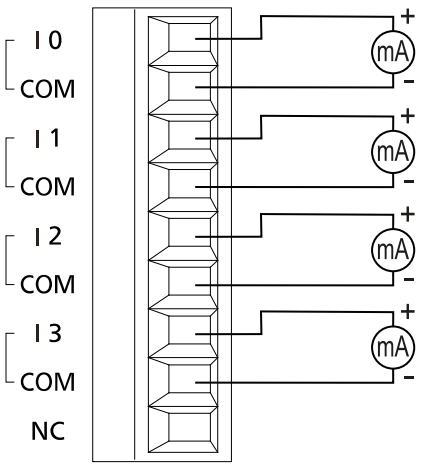
Switch 1 must be ON to use 14-bit mode. Switch 2 to 5 are used to select voltage or current output for each channel.

The DIP switch settings will become effective when the power is turned from OFF to ON.

| | Channel 0 | | Channel 1 | |
|---|---|---|---|---|
| | Voltage output | Current output | Voltage output | Current output |
|  |  |  |  |  |

| | Channel 2 | | Channel 3 | |
|---|---|---|---|---|
| | Voltage output | Current output | Voltage output | Current output |
|  |  |  |  |  |

## Output wiring

| Voltage output | Current output |
|---|---|
|  |  |

### Conversion characteristics

| -10V to +10V DC output | | -5V to +5V DC output | | 0V to 5V DC output | |
|---|---|---|---|---|---|
| **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** |
| -8000 | -10.0V | -8000 | -5.0V | 0 | 0.0V |
| -4000 | -5.0V | -4000 | -2.5V | 4000 | 1.25V |
| 0 | 0V | 0 | 0V | 8000 | 2.5V |
| +4000 | 5.0V | +4000 | +2.5V | 12000 | 3.75V |
| +8000 | 10.0V | +8000 | +5.0V | 16000 | 5.0V |

| 0V to 10V DC output | | 0mA to 20mA output | | 4mA to 20mA output | |
|---|---|---|---|---|---|
| **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** |
| 0 | 0.0V | 0 | 0.0mA | 0 | 4.0mA |
| 4000 | 2.5V | 3200 | 4.0mA | 4000 | 8.0mA |
| 8000 | 5.0V | 6400 | 8.0mA | 4000 | 12.0mA |
| 12000 | 7.5V | 9600 | 12.0mA | 4000 | 16.0mA |
| 16000 | 10.0V | 12800 | 16.0mA | 16000 | 20.0mA |
| | | 16000 | 20.0mA | | |

Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP0R Analog I/O Unit User's Manual”

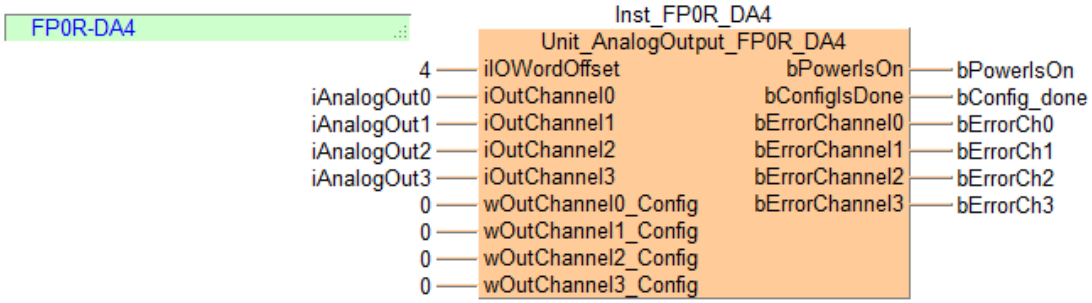
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | Inst_FP0R_DA4 | Unit_AnalogOutput_FP0R_DA4 | |
| 2 | VAR | iAnalogOut0 | INT | 0 |
| 3 | VAR | iAnalogOut1 | INT | 0 |
| 4 | VAR | iAnalogOut2 | INT | 0 |
| 5 | VAR | iAnalogOut3 | INT | 0 |
| 6 | VAR | bPowerIsOn | BOOL | FALSE |
| 7 | VAR | bConfig_done | BOOL | FALSE |
| 8 | VAR | bErrorCh0 | BOOL | FALSE |
| 9 | VAR | bErrorCh1 | BOOL | FALSE |
| 10 | VAR | bErrorCh2 | BOOL | FALSE |
| 11 | VAR | bErrorCh3 | BOOL | FALSE |

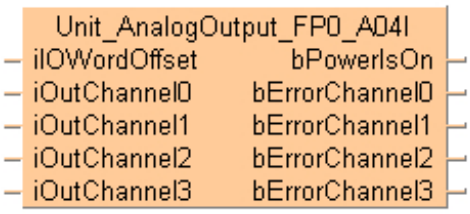
## LD body



# Unit_AnalogOutput_FP0_A04I

Function to write to an FP0-A04I unit.

This function writes digital data to the analog output channels of the analog unit. The digital values to be converted and output as analog values are specified at **iOutChannel0** to **iOutChannel3**.




## Parameters

Input

**iIOWordOffset** (INT)

Set the offset of the first WX/WY address of the analog unit according to its installation position.
For analog expansion units connected directly to the CPU (without adapter): Use **ExpansionUnitToIOWordOffset_FP0** or make the following settings: 2 (WX2/WY2) for unit number 1, 4 (WX4/WY4) for unit number 2, 6 (WX6/WY6) for unit number 3
For analog expansion units connected to the CPU via an adapter: Use **ExpansionUnitToIOWordOffset_FPX_FP0** or select the offset from the table.

| Unit position relative to the adapter | Adapter position relative to the CPU | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1st unit | 2nd unit | 3rd unit | 4th unit | 5th unit | 6th unit | 7th unit | 8th unit |
| 1st unit | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 2nd unit | 32 | 42 | 52 | 62 | 72 | 82 | 92 | 102 |
| 3rd unit | 34 | 44 | 54 | 64 | 74 | 84 | 94 | 104 |

**iOutChannel0** to **iOutChannel3** (INT)

Set the digital value to be converted and output by the analog unit.
Values: 0 to 4000

Output

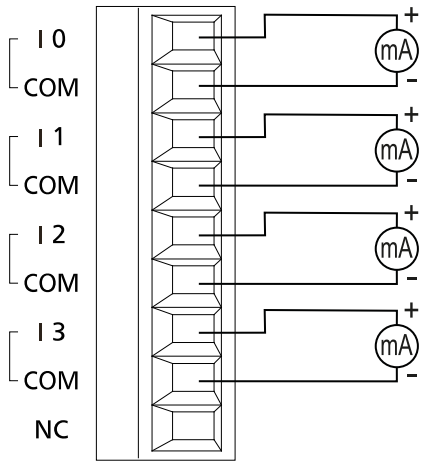
**bPowerIsOn** (BOOL)

Unit status: TRUE when the power is on.

**bErrorChannel0** to **bErrorChannel3** (BOOL)

Channel status: TRUE if there is an error.

## Output wiring




## Conversion characteristics

| Digital value (INT) | Analog value |
|---|---|
| 0 | 4.0mA |
| 500 | 6.0mA |
| 1000 | 8.0mA |
| 1500 | 10.0mA |
| 2000 | 12.0mA |
| 2500 | 14.0mA |
| 3000 | 16.0mA |
| 3500 | 18.0mA |
| 4000 | 20.0mA |

Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP0 D/A Converter Unit Technical Manual”

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

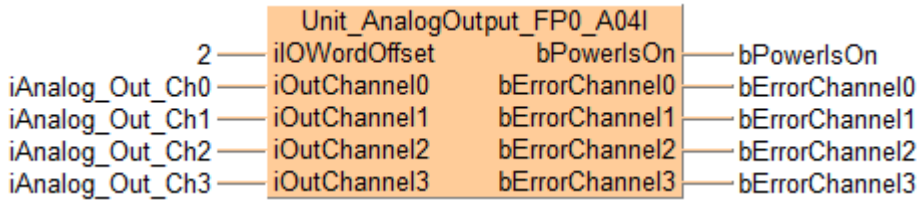
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | iAnalog_Out_Ch0 | INT | 0 |
| 2 | VAR | iAnalog_Out_Ch1 | INT | 0 |
| 3 | VAR | iAnalog_Out_Ch2 | INT | 0 |
| 4 | VAR | iAnalog_Out_Ch3 | INT | 0 |
| 5 | VAR | bPowerIsOn | BOOL | FALSE |
| 6 | VAR | bErrorChannel0 | BOOL | FALSE |
| 7 | VAR | bErrorChannel1 | BOOL | FALSE |
| 8 | VAR | bErrorChannel2 | BOOL | FALSE |
| 9 | VAR | bErrorChannel3 | BOOL | FALSE |

### POU body

The digital values entered at **iOutChannel0–iOutChannel3** are written to the analog unit and converted into analog values. The analog data is output at the corresponding output channels.

### LD body

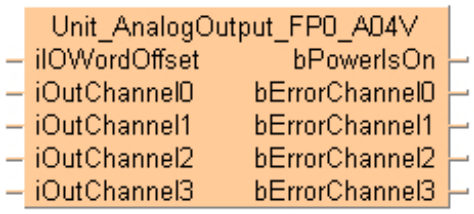
Use **ExpansionUnitNumberToIOWordOffset_FP0** or **ExpansionUnitNumberToIOWordOffset_FPX_FP0** to calculate the word offset of the analog unit connected to the CPU.



# Unit_AnalogOutput_FP0_A04V

Function to write to an FP0-A04V unit.

This function writes digital data to the analog output channels of the analog unit. The digital values to be converted and output as analog values are specified at **iOutChannel0** to **iOutChannel3**.




## Parameters

Input

**iIOWordOffset** (INT)

Set the offset of the first WX/WY address of the analog unit according to its installation position.
For analog expansion units connected directly to the CPU (without adapter): Use **ExpansionUnitToIOWordOffset_FP0** or make the following settings: 2 (WX2/WY2) for unit number 1, 4 (WX4/WY4) for unit number 2, 6 (WX6/WY6) for unit number 3
For analog expansion units connected to the CPU via an adapter: Use **ExpansionUnitToIOWordOffset_FPX_FP0** or select the offset from the table.

| Unit position relative to the adapter | Adapter position relative to the CPU | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1st unit | 2nd unit | 3rd unit | 4th unit | 5th unit | 6th unit | 7th unit | 8th unit |
| 1st unit | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 2nd unit | 32 | 42 | 52 | 62 | 72 | 82 | 92 | 102 |
| 3rd unit | 34 | 44 | 54 | 64 | 74 | 84 | 94 | 104 |

**iOutChannel0** to **iOutChannel3** (INT)

Set the digital value to be converted and output by the analog unit.
Values: -2000 to +2000

Output

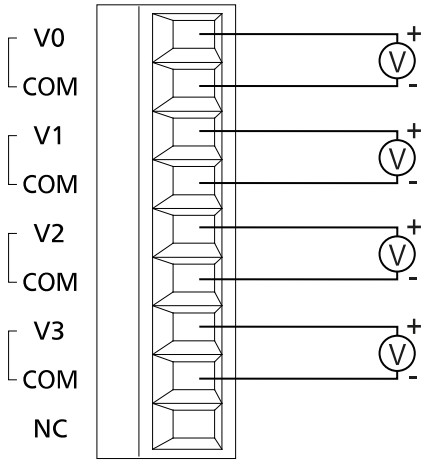
**bPowerIsOn** (BOOL)

Unit status: TRUE when the power is on.

**bErrorChannel0** to **bErrorChannel3** (BOOL)

Channel status: TRUE if there is an error.

## Output wiring




## Conversion characteristics

| Digital value (INT) | Analog value |
|---|---|
| -2000 | -10.0V |
| -1500 | -7.5V |
| -1000 | -5.0V |
| -500 | -2.5V |
| 0 | 0.0V |
| +500 | +2.5V |
| +1000 | +5.0V |
| +1500 | +7.5V |
| +2000 | +10.0V |

Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP0 D/A Converter Unit Technical Manual”

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

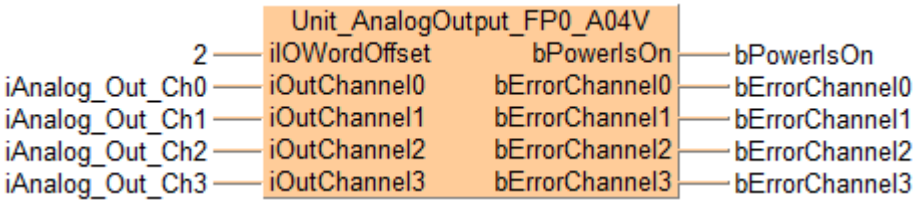
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | iAnalog_Out_Ch0 | INT | 0 |
| 2 | VAR | iAnalog_Out_Ch1 | INT | 0 |
| 3 | VAR | iAnalog_Out_Ch2 | INT | 0 |
| 4 | VAR | iAnalog_Out_Ch3 | INT | 0 |
| 5 | VAR | bPowerIsOn | BOOL | FALSE |
| 6 | VAR | bErrorChannel0 | BOOL | FALSE |
| 7 | VAR | bErrorChannel1 | BOOL | FALSE |
| 8 | VAR | bErrorChannel2 | BOOL | FALSE |
| 9 | VAR | bErrorChannel3 | BOOL | FALSE |

### POU body

The digital values entered at **iOutChannel0–iOutChannel3** are written to the analog unit and converted into analog values. The analog data is output at the corresponding output channels.

### LD body

Use **ExpansionUnitNumberToIOWordOffset_FP0** or **ExpansionUnitNumberToIOWordOffset_FPX_FP0** to calculate the word offset of the analog unit connected to the CPU.

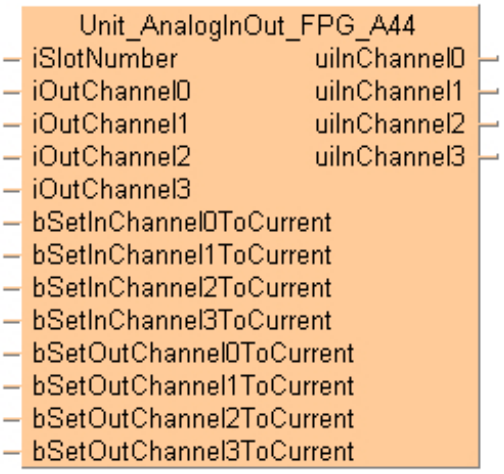


# Unit_AnalogInOut_FPG_A44

Function to write to and read from FPG-A44 unit

This function writes digital data to the analog output channels of the analog unit and reads converted digital data from its analog input channels. The digital values to be converted and output as analog values are specified at **iOutChannel0** to **iOutChannel3**. The converted digital values from the analog unit are stored per channel in the output variables **iInChannel0** to **iInChannel3**.

The input and output type (voltage or current) is also set with this function.




## Parameters

Input

**iSlotNumber** (INT)

Specify the slot number.

**iOutChannel0** to **iOutChannel3** (INT)

Set the digital value to be converted and output by the analog unit.

**bSetInChannel0ToCurrent** to **bSetInChannel3ToCurrent** (BOOL)

Set voltage or current input for each channel number.

- TRUE: Current input
- FALSE: Voltage input

**bSetOutChannel0ToCurrent** to **bSetOutChannel3ToCurrent** (BOOL)

Set voltage or current output for each channel number.

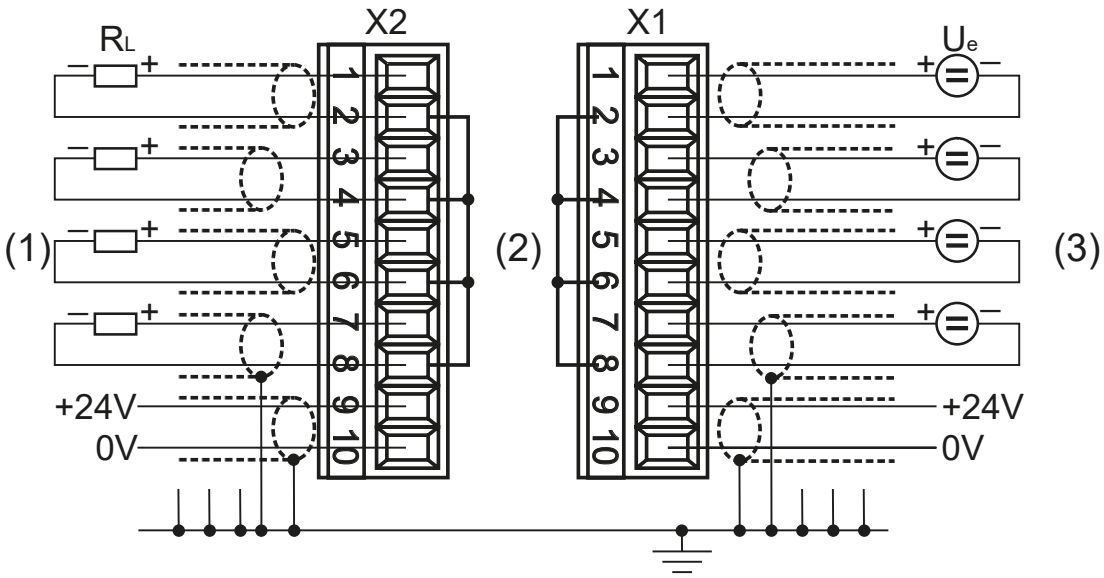
- TRUE: Current output
- FALSE: Voltage output

Output

**uiInChannel0** to **uiInChannel3** (UINT)

Returns the converted digital data from the analog unit by channel.

## Wiring diagram




(1) Output
(2) Internal connection
(3) Input

## D/A conversion

| Voltage output | | Current output | |
|---|---|---|---|
| **Digital value (INT)** | **Analog value** | **Digital value (INT)** | **Analog value** |
| 4095 | 10V | 4095 | 20mA |
| 2048 | 5V | 2048 | 12mA |
| 0 | 0V | 0 | 4mA |
| -2048 | -5V | | |
| -4095 | -10V | | |

### A/D conversion

| Voltage input | | Current input | |
|---|---|---|---|
| **Digital value (UINT)** | **Analog value** | **Digital value (UINT)** | **Analog value** |
| 65535 | 10V | 65535 | 20mA |
| 32768 | 5V | 39321 | 12mA |
| 0 | 0V | 13107 | 4mA |
| | | 0 | 0 |

## Example

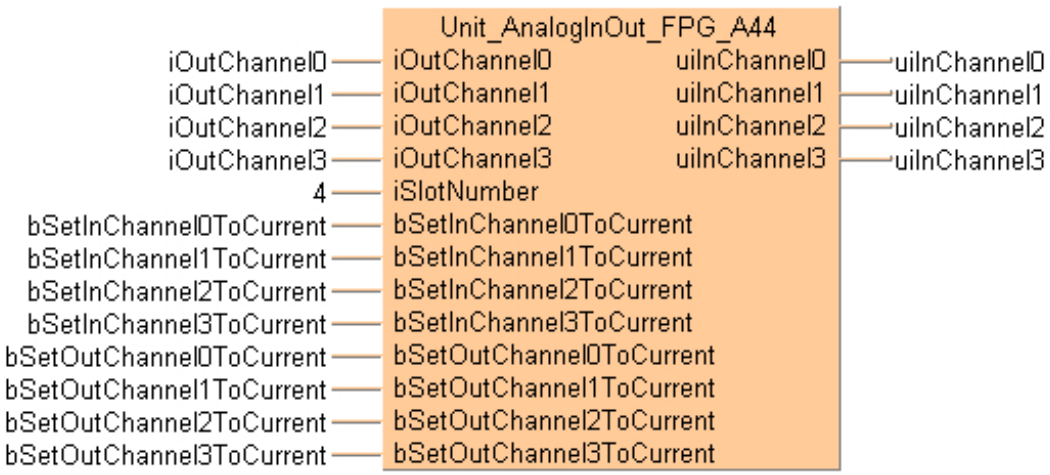
### POU header

All input and output variables used for programming this function have been declared in the POU header.

The same POU header is used for all programming languages.

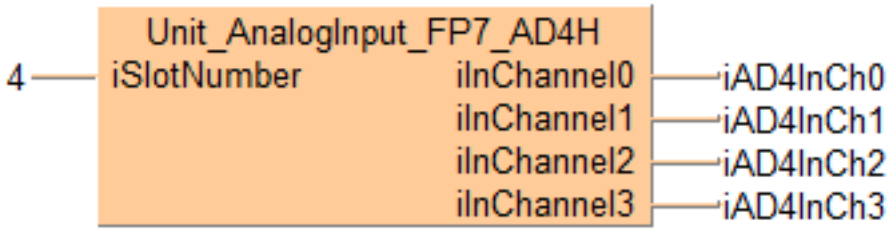
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iOutChannel0 | INT | 0 |
| 1 | VAR | iOutChannel1 | INT | 0 |
| 2 | VAR | iOutChannel2 | INT | 0 |
| 3 | VAR | iOutChannel3 | INT | 0 |
| 4 | VAR | bSetInChannel0ToCurrent | BOOL | FALSE |
| 5 | VAR | bSetOutChannel0ToCurrent | BOOL | FALSE |
| 6 | VAR | bSetInChannel1ToCurrent | BOOL | FALSE |
| 7 | VAR | bSetInChannel2ToCurrent | BOOL | FALSE |
| 8 | VAR | bSetInChannel3ToCurrent | BOOL | FALSE |
| 9 | VAR | bSetOutChannel1ToCurrent | BOOL | FALSE |
| 10 | VAR | bSetOutChannel2ToCurrent | BOOL | FALSE |
| 11 | VAR | bSetOutChannel3ToCurrent | BOOL | FALSE |
| 12 | VAR | uiInChannel0 | UINT | 0 |
| 13 | VAR | uiInChannel1 | UINT | 0 |
| 14 | VAR | uiInChannel2 | UINT | 0 |
| 15 | VAR | uiInChannel3 | UINT | 0 |

### LD body



# Unit_AnalogInput_FP7_AD4H

Function to read from an FP7-AD4H unit.




This function reads converted digital data from the analog input channel of the analog unit in the slot specified by **iSlotNumber**. The converted digital values are stored per channel in the output variables **iInChannel0** to **iInChannel3**.

## Parameters

Input

**iSlotNumber** (INT)

Set the slot number of the analog unit. The FP7 CPU has slot number 0.

Output

**iInChannel0** to **iInChannel3** (INT)

Returns the converted digital data from the analog unit by channel.
Values:
For -10 to +10V: -31250 to +31250
For 0 to 10V, 0 to 5V, 0 to 20 mA: 0 to 31250
For 1 to 5 V, 4 to 20 mA: 0 to 25000

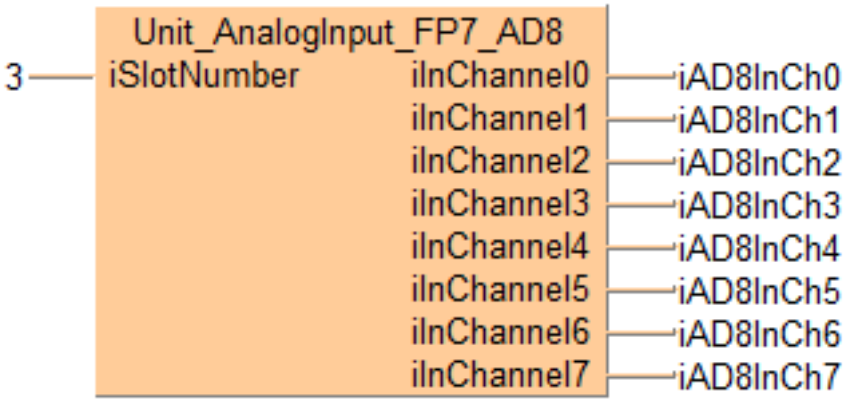
### Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP7 Analog Input Unit User's Manual”

# Unit_AnalogInput_FP7_AD8

Function to read from an FP7-AD8 unit.




This function reads converted digital data from the analog input channel of the analog unit in the slot specified by **iSlotNumber**. The converted digital values are stored per channel in the output variables **iInChannel0** to **iInChannel7**.

## Parameters

Input

**iSlotNumber** (INT)

Set the slot number of the analog unit. The FP7 CPU has slot number 0.

Output

**iInChannel0** to **iInChannel7** (INT)

Returns the converted digital data from the analog unit by channel.
Values:

For -10 to +10V: -31250 to +31250
For 0 to 10V, 0 to 5V, 0 to 20 mA: 0 to 31250
For 1 to 5 V, 4 to 20 mA: 0 to 25000

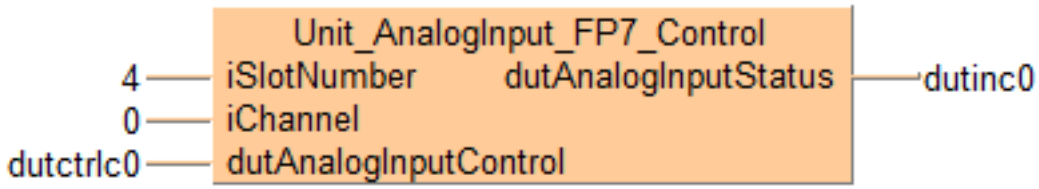
### Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP7 Analog Input Unit User's Manual”

# Unit_AnalogInput_FP7_Control

Function to write control values to and read status values from an FP7 analog unit.




This function reads the control values specified in the DUT at **dutAnalogInputControl** for the selected channel and outputs status values at **dutAnalogInputStatus**. The function is optional and can be used to override the settings that are made in the “I/O map and unit configuration” dialog of Control FPWIN Pro7.

## Parameters

Input

**iSlotNumber** (INT)

Set the slot number of the analog unit. The FP7 CPU has slot number 0.

**iChannel** (INT)

Set the channel number.

**dutAnalogInputControl** (Unit_AnalogInput_FP7_Control_WY_DUT)

Select the DUT from the FP tool library.

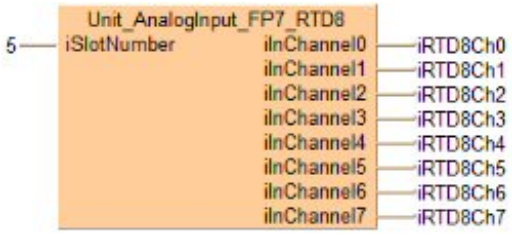
Output

**dutAnalogInputStatus** (Unit_AnalogInput_FP7_Status_WX_DUT)

Select the DUT from the FP tool library.

# Unit_AnalogInput_FP7_RTD8

Function to read from an FP7-RT8D unit.




This function reads converted digital data from the analog input channel specified by **iSlotNumber**. The converted digital values are stored per channel in the output variables **iInChannel0** to **iInChannel7**.

## Parameters

Input

**iSlotNumber** (INT)

Set the slot number of the analog unit. The FP7 CPU has slot number 0.

Output

**iInChannel0** to **iInChannel7** (INT)

Returns the converted digital data from the analog unit by channel.

## Conversion characteristics

These characteristics apply for units manufactured in November 2016 or later (lot numbers 161100 or higher). Refer to the manual for the characteristics of older units.

- Pt100, JPt100

| Range 1 | | Range 2 | |
|---|---|---|---|
| **Analog value (°C)** | **Digital value** | **Analog value (°C)** | **Digital value** |
| -130.0 or less | +30000 | -230.0 or less | +30000 |
| -130 to -115 | -1150 | -230 to -115 | -1500 |
| +215 to +230 | +1150 | +665 to +680 | +6650 |
| +230.0 or more | +30000 | +680.0 or more | +30000 |

- Pt1000

| Range 1 | |
|---|---|
| **Analog value (°C)** | **Digital value** |
| -120.0 or less | -300/*-220* |
| -120 to -115 | +1500/+3020 |
| +115 to +120 | +1150 |
| +120.0 or more | +30000 |

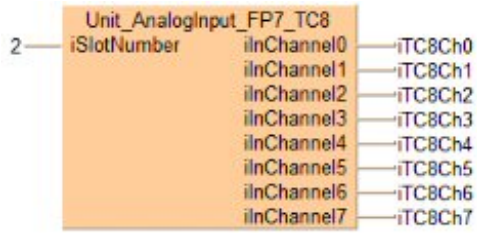
Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP7 Thermocouple Multi-analog Input Unit/FP7 RTD Input Unit User's Manual”

# Unit_AnalogInput_FP7_TC8

Function to read from an FP7-TC8 unit.




This function reads converted digital data from the analog input channel specified by **iSlotNumber**. The converted digital values are stored per channel in the output variables **iInChannel0** to **iInChannel7**.

## Parameters

Input

**iSlotNumber** (INT)

Set the slot number of the analog unit. The FP7 CPU has slot number 0.

Output

**iInChannel0**–**iInChannel7** (INT)

Returns the converted digital data from the analog unit by channel.
Values:
Voltage and current input:
For -10 to +10V: -31250 to +31250
For 0 to 10V, 0 to 5V, 0 to 20 mA: 0 to 31250
For 1 to 5 V, 4 to 20 mA: 0 to 25000

Thermocouple input:

| Type | Analog value | Digital value (INT) |
|---|---|---|
| K1 | -115°C or less | -1150 |
| | +615°C or more | +6150 |
| K2 | -215°C or less | -2150 |
| | +1385°C or more | +13850 |
| J1 | -115°C or less | -1150 |
| | +415°C or more | +4150 |
| J2 | -215°C or less | -2150 |
| | + 1215°C or more | +12150 |
| T | -285°C or less | -2850 |

| Type | Analog value | Digital value (INT) |
|---|---|---|
| | +415°C or more | + 4150 |
| N | -285°C or less | -2850 |
| | + 1315°C or more | + 13150 |
| R | -15°C or less | -150 |
| | +1775°C or more | +17750 |
| S | -15°C or less | -150 |
| | +1775°C or more | + 17750 |
| B | -15°C or less | -150 |
| | 1835°C or more | + 18350 |
| E | -285°C or less | -2850 |
| | +1015°C or more | + 10150 |
| PL | -15°C or less | -150 |
| | +1405°C or more | + 14050 |
| WRe5-26 | -15°C or less | -150 |
| | + 2330°C or more | + 23300 |

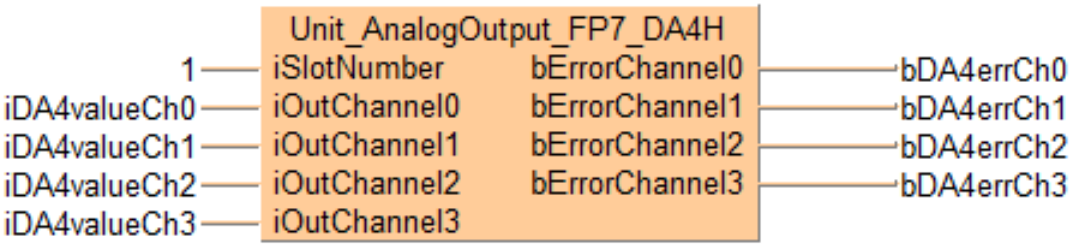
Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP7 Thermocouple Multi-analog Input Unit/FP7 RTD Input Unit User's Manual”

# Unit_AnalogOutput_FP7_DA4H

Function block to write to an FP7-DA4H unit.




This function writes digital data to the analog output channels of the analog unit. The digital values to be converted and output as analog values are specified at **iOutChannel0** to **iOutChannel3**.

## Parameters

Input

**iSlotNumber** (INT)

Set the slot number of the analog unit. The FP7 CPU has slot number 0.

Output

**iOutChannel00** to **iOutChannel03** (INT)

Set the digital value to be converted and output by the analog unit.
Values:

For -10 to +10V: -31250 to +31250
For 0 to 10V, 0 to 5V, 0 to 20 mA: 0 to 31250
For 1 to 5 V, 4 to 20 mA: 0 to 25000

**bErrorChannel0** to **bErrorChannel3** (BOOL)

Channel status: TRUE if there is an error.

### Tip

This command description provides basic hardware documentation only. For detailed technical information, consult the manual:

“FP7 Analog Output Unit User's Manual”

## 21.22 FP instructions

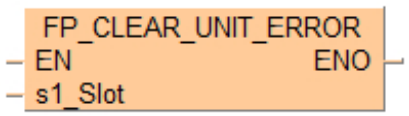
Tip

Advantages of FP instructions

# FP_CLEAR_UNIT_ERROR

Clear error/warning of units

This FP instruction clears an error or warning in the unit attached to the slot number specified by **s1_Slot** if the trigger **EN** is TRUE. The instruction can be used for the following types of units: High-speed counter unit, positioning unit, pulse output unit, motion control unit, serial communication unit.




## Parameters

Input

**s1_Slot** (WORD, INT, UINT)

Slot number

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the slot number is outside the permissible range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the slot number is outside the permissible range

## Example

## POU header

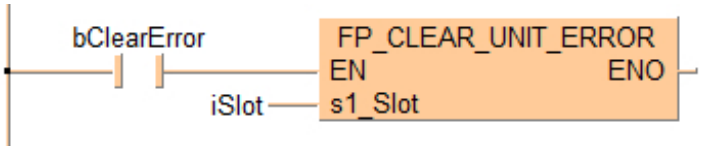
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bClearError | BOOL | FALSE |
| 1 | VAR | iSlot | INT | 1 |

### POU body

When the variable **bClearError** is set to TRUE, the instruction is carried out.

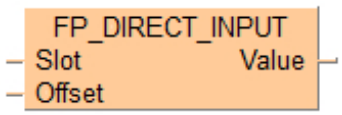
### LD body



# FP_DIRECT_INPUT

Direct input refresh

This FP instruction reads the current value of the direct input memory specified by **Slot** and **Offset** independently from any IO refresh.




## Parameters

Input

**Slot** (WORD, INT, UINT)

Slot number

**Offset** (WORD, INT, UINT)

Offset

Output

**Value** (BOOL, WORD, DWORD)

Value of the direct input

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

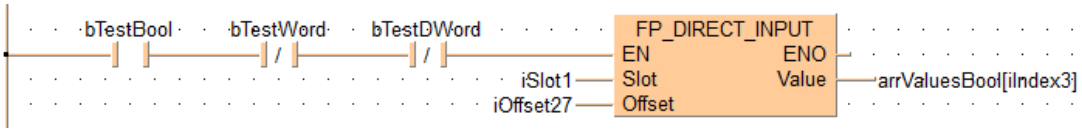
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bTestBool | BOOL | FALSE |
| 1 | VAR | bTestWord | BOOL | FALSE |
| 2 | VAR | bTestDWord | BOOL | FALSE |
| 3 | VAR | iOffset27 | INT | 27 |
| 4 | VAR | iSlot1 | INT | 1 |
| 5 | VAR | iIndex3 | INT | 3 |
| 6 | VAR | arrValuesBool | ARRAY [0..15] of BOOL | [3(FALSE),TRUE,12(FALSE)] |

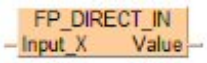
## LD body



# FP_DIRECT_IN

Read direct input flag

This instruction reads the value of an input flag (X) using a direct execution command. The **Input_X** will be read asynchronously within the current PLC cycle.




Input

**Input_X** FP0H, FP-XH: (BOOL); FP7: (BOOL, WORD, DWORD)

Global variable with FP address or explicit user address to be read directly.
FP0H, FP-XH only: Valid range for FP address is X0–X109F to which a real input pin is assigned. Otherwise the PLC goes into operation error.

Output

**Value** FP0H, FP-XH: (BOOL); FP7: (BOOL, WORD, DWORD)

Value read directly from **Input_X**

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

| | Class | Identifier | FP address | IEC address | Type | Initial |
|---|---|---|---|---|---|---|
| 1 | VAR_GLOBAL | g_bDirectIn_X1 | X1 | %IX0.1 | BOOL | FALSE |
| 2 | VAR_GLOBAL | g_bDirectOut_Y1 | Y1 | %QX0.1 | BOOL | FALSE |

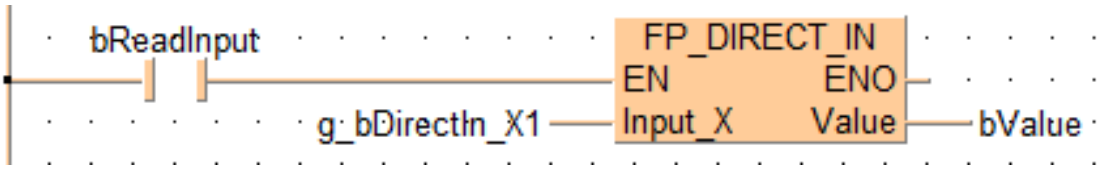
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bValue | BOOL | FALSE |
| 2 | VAR_EXTERNAL | g_bDirectIn_X1 | BOOL | FALSE |
| 3 | VAR | bReadInput | BOOL | FALSE |

### LD body

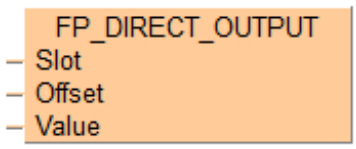
When the variable **bReadInput** is set to TRUE, the function is carried out.



# FP_DIRECT_OUTPUT

Direct output refresh

This FP instruction sets the direct output memory specified by **Slot** and **Offset** independently from any IO refresh. Additionally, it sets also the corresponding output flag or flags.




## Parameters

Input

**Slot** (WORD, INT, UINT)

Slot number

**Offset** (WORD, INT, UINT)

Offset

**Value** (BOOL, WORD, DWORD)

Value of the direct output

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

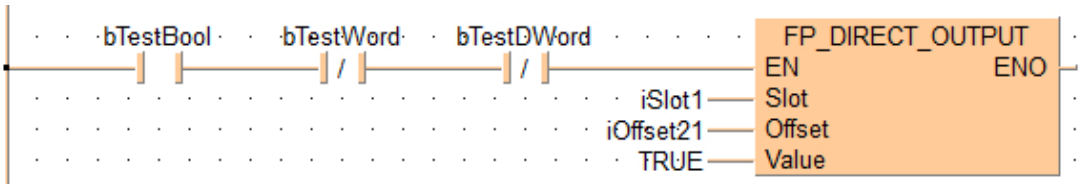
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bTestBool | BOOL | FALSE |
| 1 | VAR | bTestWord | BOOL | FALSE |
| 2 | VAR | bTestDWord | BOOL | FALSE |
| 3 | VAR | iSlot1 | INT | 1 |
| 4 | VAR | iOffset21 | INT | 21 |

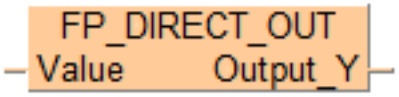
**LD body**



# FP_DIRECT_OUT

Write direct output flag

This instruction writes the value of an output flag (Y) using a direct execution command. The **Output_Y** will be written asynchronously within the current PLC cycle.




Input

**Value** FP0H, FP-XH: (BOOL); FP7: (BOOL, WORD, DWORD)

Value written directly to **Output_Y**

Output

**Output_Y** FP0H, FP-XH: (BOOL); FP7: (BOOL, WORD, DWORD)

Global variable with FP address or explicit user address to be written directly.
FP0H, FP-XH only: Valid range for FP address is Y0–Y109F to which a real output pin is assigned. Otherwise the PLC goes into operation error.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

| | Class | Identifier | FP address | IEC address | Type | Initial |
|---|---|---|---|---|---|---|
| 1 | VAR_GLOBAL | g_bDirectIn_X1 | X1 | %IX0.1 | BOOL | FALSE |
| 2 | VAR_GLOBAL | g_bDirectOut_Y1 | Y1 | %QX0.1 | BOOL | FALSE |

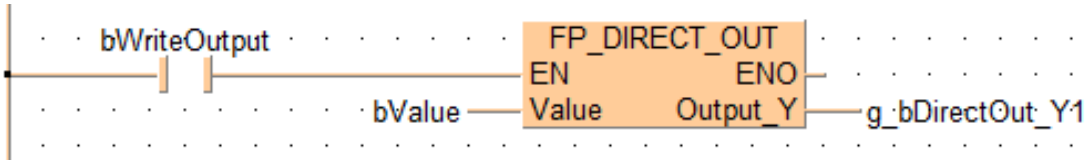
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bWriteOutput | BOOL | FALSE |
| 2 | VAR_EXTERNAL | g_bDirectOut_Y1 | BOOL | FALSE |
| 3 | VAR | bValue | BOOL | FALSE |

### LD body

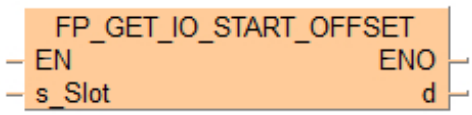
When the variable **bWriteOutput** is set to TRUE, the function is carried out.



# FP_GET_IO_START_OFFSET

Get offset of I/O address

This FP instruction gets the start offset of an I/O address of the slot specified by **s_Slot** and stores the result in **d**.




## Parameters

Input

**s_Slot** (WORD, INT, UINT)

Slot number: 0–64

Output

**d** (DWORD, DINT, UDINT, DATE, TOD, DT)

Offset for the first valid I/O address

## Remarks

We recommend using **FP_GET_UNIT_OFFSETS1** to get all offsets from DIX, DIY and DI2 in a standardized way that allows to access all relevant memories of a unit.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

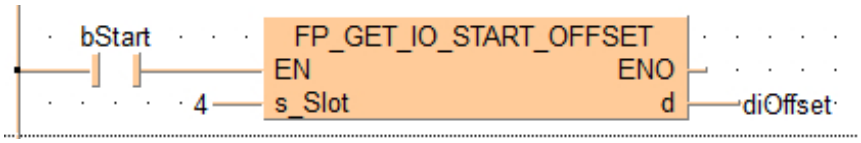
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | diOffset | DINT | 0 |

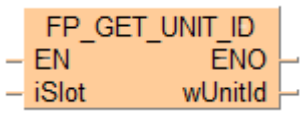
**LD body**



# FP_GET_UNIT_ID

returns the expansion unit ID of a specified slot

This FP instruction returns the unit ID of the expansion unit of the slot number specified by **iSlot**. The unit ID found in “I/O map and unit configuration” is written to the output variable **wUnitId** and can be used for comparison with a system variable of the group “PLC unit constants”.




Input

**iSlot** (WORD, INT, UINT)

Slot number of expansion unit

Output

**wUnitId** (WORD, INT, UINT)

Unit ID of expansion unit, e.g. **SYS_UNIT_AFP7X16DW**

## Remarks

Before using this instruction, please re-compile your PLC program and download it with “Online” > “Download program code and PLC configuration to PLC” to the PLC. This prevents a wrong unit ID to be returned in case the unit configuration has been changed.

## Example

## POU header

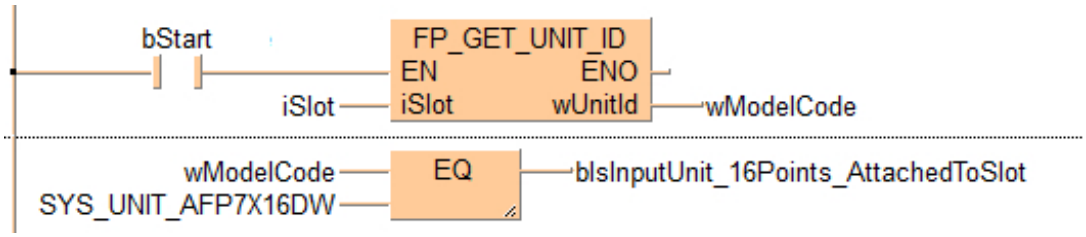
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | iSlot | INT | 1 |
| 2 | VAR | wModelCode | WORD | 0 |
| 3 | VAR | bIsInputUnit_16Points_AttachedToSlot | BOOL | FALSE |

## POU body

When the variable **bStart** is set to TRUE, the function is carried out.

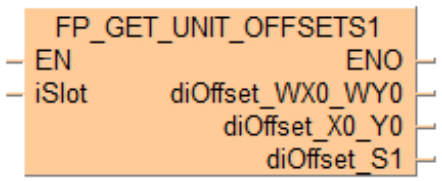
### LD body


bStart
FP_GET_UNIT_ID
EN
ENO
iSlot
iSlot
wUnitId
wModelCode
wModelCode
SYS_UNIT_AFP7X16DW
EQ
bIsInputUnit_16Points_AttachedToSlot

# FP_GET_UNIT_OFFSETS1

Calculate the I/O offset of an expansion unit

This instruction calculates the relevant offset values of an expansion unit in the slot number specified by **iSlot** for accessing the input/output or unit memories via index modifiers.




## Parameters

Input

**iSlot** (WORD, INT, UINT)

Slot number

Output

**diOffset_WX0_WY0** (DWORD, DINT, UDINT, DATE, TOD, DT)

The starting I/O word number

**diOffset_X0_Y0** (DWORD, DINT, UDINT, DATE, TOD, DT)

The starting I/O word number multiplied by 16

**diOffset_S1** (DWORD, DINT, UDINT, DATE, TOD, DT)

Slot number minus 1

## Remarks

**iSlot** = 5 and starting word number = 16

| Starting word number | Corresponding address | Application example | Explanation |
|---|---|---|---|
| **diOffset_WX0_WY0** = 16 | DIX | DIXWX1 or DIXWY1 | Access to I/O word 1 of the unit in slot 5 |
| **diOffset_X0_Y0** = 16*16=256 | DIY | DIYX12 or DIYY12 | Access to I/O bit hex 12 of the unit in slot 5 |
| **diOffset_S1** = 4 | DI2 | DI2S1:UM1A | Access to unit memory word hex 1A of the unit in slot 5 |

## Example

### POU header

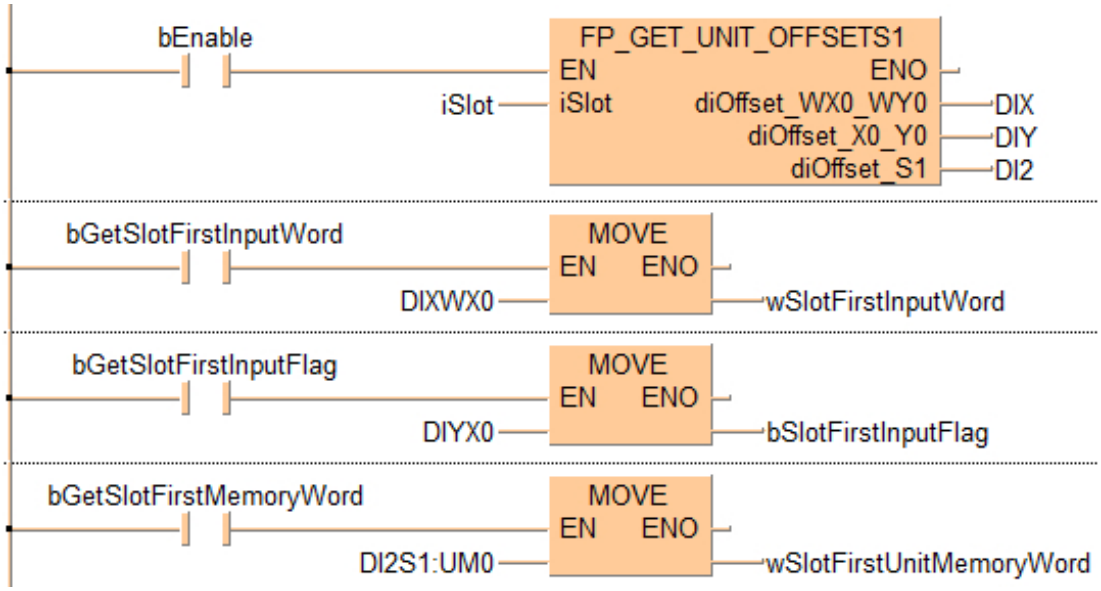
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iSlot | INT | 1 |
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | wSlotFirstInputWord | WORD | 0 |
| 3 | VAR | bSlotFirstInputFlag | BOOL | FALSE |
| 4 | VAR | wSlotFirstUnitMemoryWord | WORD | 0 |
| 5 | VAR | bGetSlotFirstInputWord | BOOL | FALSE |
| 6 | VAR | bGetSlotFirstInputFlag | BOOL | FALSE |
| 7 | VAR | bGetSlotFirstMemoryWord | BOOL | FALSE |

### POU body

When the variable **bEnable** is set to TRUE, the function is carried out.

### LD body


bEnable
FP_GET_UNIT_OFFSETS1
EN
ENO
iSlot
iSlot
diOffset_WX0_WY0
DIX
diOffset_X0_Y0
DIY
diOffset_S1
DI2
bGetSlotFirstInputWord
MOVE
EN
ENO
DIXWX0
wSlotFirstInputWord
bGetSlotFirstInputFlag
MOVE
EN
ENO
DIYX0
bSlotFirstInputFlag
bGetSlotFirstMemoryWord
MOVE
EN
ENO
DI2S1:UM0
wSlotFirstUnitMemoryWord

## 21.23 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

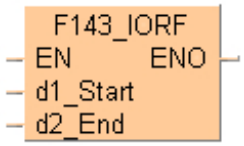
Advantages of IEC instructions

Advantages of FP instructions

# F143_IORF

Partial I/O update

Updates the inputs or outputs specified by the value of **d1_Start** (starting word address) and the value of **d2_End** (ending word address) immediately after the trigger **EN** is in the ON-state even in the program execution stage.




## Parameters

Input

**d1_Start** (WORD, INT, UINT)

starting word address

**d2_End** (WORD, INT, UINT)

ending word address

## Remarks

Using this instruction, you can update inputs or outputs without the time-lag caused by scanning.

The same type of operand should be specified for **d1_Start** and **d2_End**.

### PLCs with configurable, serially numbered I/O addresses:

FP2, FP2SH, FP3 /5 /10 /10SH (PLCs **with** backplanes)

- Specify the word address as 0 ≤**d1_Start**≤**d2_End**≤ 127.If only WX10 (or WY10) are to be updated based on the I/O-address configuration, **d1_Start** and **d2_End** will be set as follows: **d1_Start** = 10 and **d2_End** = 10.
- Set the same word address in **d1_Start** and **d2_End** to update only 1 word.

The partial I/O update instruction is executed only for the I/O units on the master backplane or expansion backplane. It is not executed for the I/O unit in the slave station of the Remote I/O System.

### PLCs whose I/O addresses cannot be configured and are not serially numbered:

FP-Σ, FP0 (PLCs **without** backplanes)

The instruction **F143_IORF** updates the inputs and outputs specified by **d1_Start** (starting word address) and **d2_End** (ending word address) immediately after the trigger turns ON even in the program execution stage.

Note

- With the FP0 and FP-Σ, refreshing initiated by the IORF command is done only for the control unit.
- If **d1_Start** and **d2_End** are variables and not constants, then the compiler automatically accesses the variables' values via the index register.
- The same type of operand should be specified for **d1_Start** and **d2_End**.
- With input refreshing, WX0 should be specified for **d1_Start** and **d2_End**.
- With output refreshing, WY0 should be specified for **d1_Start** and **d2_End**.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

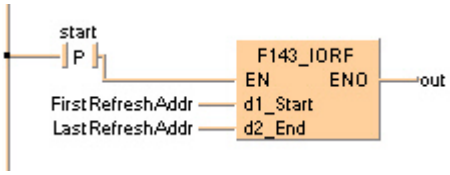
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the funtion |
| 1 | VAR | FirstRefreshAddr | INT | 10 | |
| 2 | VAR | LastRefreshAddr | INT | 10 | |

### POU body

When the variable **start** changes from FALSE to TRUE, the function is carried out.

To update WX10 and WY10 based on the master I/O map configuration, set d1 = 10 and d2 = 10.

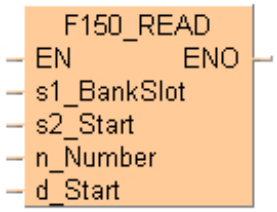
### LD body



# F150_READ

Data read from intelligent units

Reads data from the shared memory in an intelligent module.




## Parameters

Input

**s1_BankSlot** (WORD, INT, UINT)

Specifies the bank/slot number in the shared memory of the intelligent module

**s2_Start** (WORD, INT, UINT)

Specifies the starting address in the shared memory of the intelligent module (source data address)

**n_Number** (INT)

Specifies the number of words to be read

**d_Start** (WORD, INT, UINT)

Starting address in the CPU for storing data read (destination address)

## Remarks

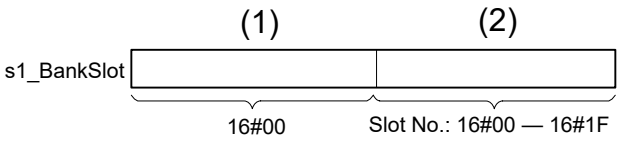
The **n_Number** words of the data stored in the shared memory of the intelligent unit/board specified by **s1_BankSlot** are read from the address specified by **s2_Start**, and are stored in the area specified by **d_Start** of the CPU.

The number of variable arguments at the inputs is limited by the available index registers of the PLC.

Specifying **s1_BankSlot**

- Intelligent unit without bank

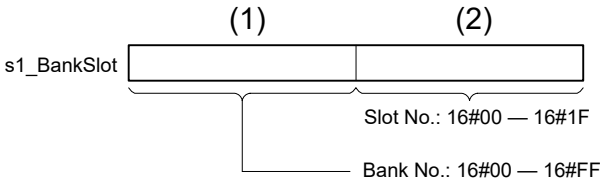
  Specify the slot number in which the target intelligent unit has been installed.




(1) Upper byte
(2) Lower byte

- Intelligent unit with bank

  Specify the slot number (hex. constant) in which the target intelligent unit has been installed, and the bank number (hex. constant).




(1) Upper byte
(2) Lower byte

| Reference: | Intelligent unit with bank | |
|---|---|---|
| | Name | Order Number |
| | FP3 expansion data memory unit | AFP32091<br>AFP32092 |
| | FPΣ expansion data memory unit | AFPG201 |

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if **s1_BankSlot** exceeds the limit of specified range
- if the data read exceeds the area of **d_Start**

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if **s1_BankSlot** exceeds the limit of specified range
- if the data read exceeds the area of **d_Start**

## Example

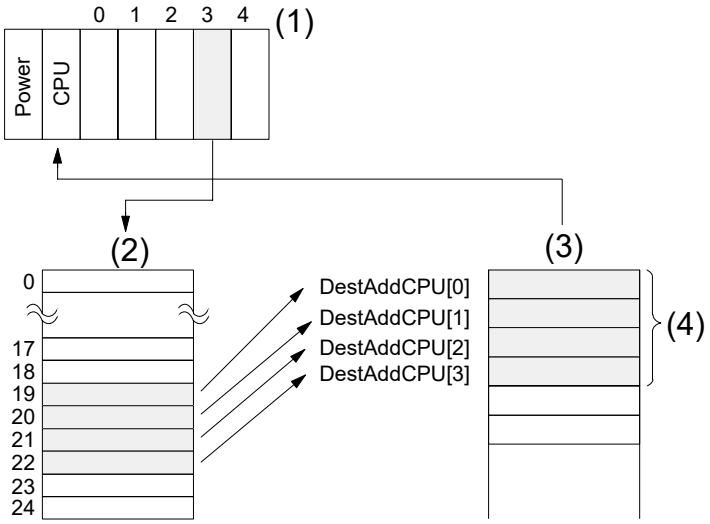
## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | Start | BOOL | FALSE | activates the function |
| 1 | VAR | Slot No | WORD | 16#03 | if start is TRUE, this value |
| 2 | VAR | Addr Data To Read | INT | 19 | Starting address in intelligent |
| 3 | VAR | No Words To Read | INT | 4 | |
| 4 | VAR | Dest Addr CPU | ARRAY [0..3] OF INT | [4(0)] | Starting address in CPU to store data read |
| 5 | VAR | | | | |

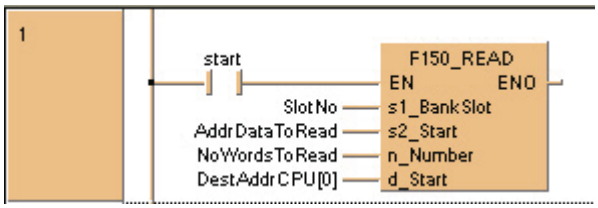
## POU body

Reads 4 words of data stored in the addresses starting from 19, specified in **AddrDataToRead**, of the intelligent unit's shared memory (located in slot 3). Then it stores them in the array **DestAddrCPU**, when **Start** turns on.




(1) Slot Number
(2) Intelligent unit
(3) CPU
(4) 4 words

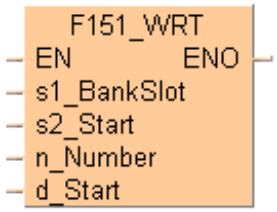
## LD body



# F151_WRT

Write into memory of intelligent units

Writes data into the shared memory of an intelligent unit.




## Parameters

Input

**s1_BankSlot** (WORD, INT, UINT)

Specifies the bank/slot number in the shared memory of the intelligent module

**s2_Start** (WORD, INT, UINT)

Starting address for data in the shared memory of the CPU

**n_Number** (INT)

Specifies the number of words to be written to the shared memory

**d_Start** (WORD, INT, UINT)

Specifies the starting address in the intelligent unit for storing data written (destination address)

## Remarks

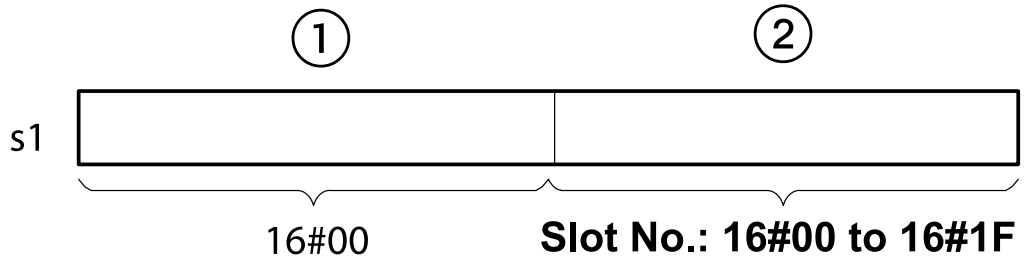
Writes **n_Number** words of the initial data from the area specified by **s2_Start** of the CPU to the address specified by **d_Start** of the shared memory of the intelligent unit specified by **s1_BankSlot**.

The number of variable arguments at the inputs is limited by the available index registers of the PLC.

Specifying **s1_BankSlot**

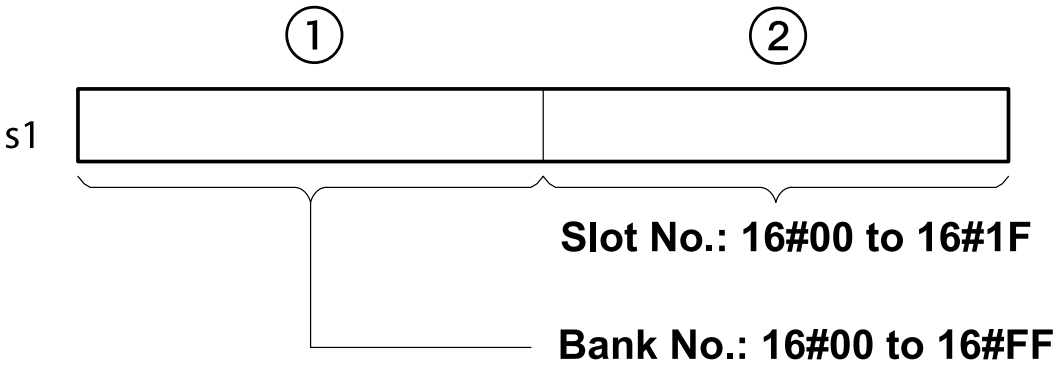
- Intelligent unit without bank

  Specify the slot number in which the target intelligent unit has been installed.




- Intelligent unit with bank

  Specify the slot number (hex. constant) in which the target intelligent unit has been installed, and the bank number (hex. constant).




| Reference: | Intelligent unit with bank | |
|---|---|---|
| | Name | Order Number |
| | FP3 expansion data memory unit | AFP32091<br>AFP32092 |
| | FPΣ expansion data memory unit | AFPG201 |

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if **s1_BankSlot** exceeds the limit of specified range
- if the data read exceeds the area of **d**

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if **s1_BankSlot** exceeds the limit of specified range
- if the data read exceeds the area of **d**

## Example

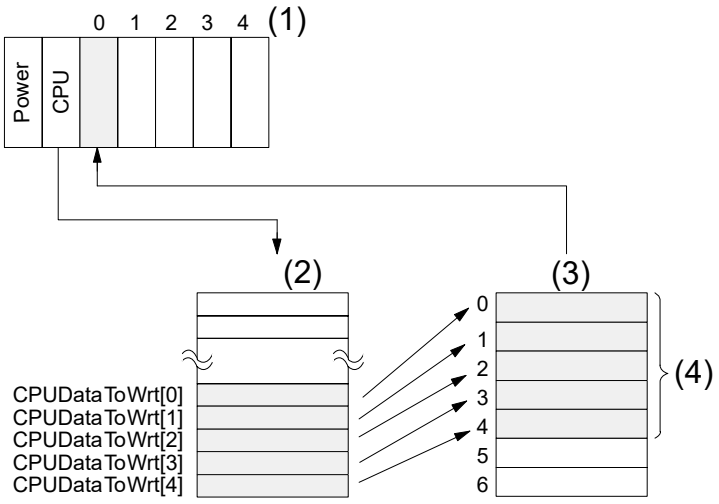
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | Start | BOOL | FALSE | activates the function |
| 1 | VAR | Slot No | WORD | 16#00 | if start is TRUE, this value |
| 2 | VAR | CPU Data To Wrt | ARRAY [0..4] OF INT | [5,10,15,20,25] | |
| 3 | VAR | No Words To Write | INT | 5 | |
| 4 | VAR | Destination Addr | INT | 0 | Starting 16-bit address for storing data in the intelligent unit |
| 5 | VAR | | | | |

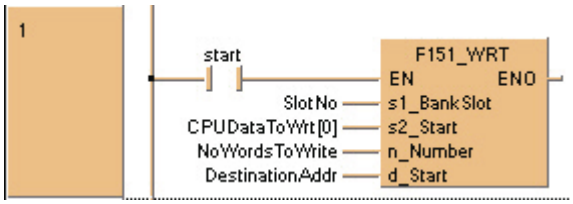
### POU body

Five words of data defined in **CPUDataToWrt** are written into the addresses starting from 0 to 4 of the intelligent unit's shared memory (located in slot 0) when Start turns on.




1. Slot No.
2. CPU
3. Intelligent unit
4. 5 words

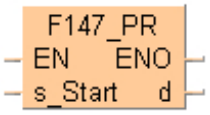
### LD body



# F147_PR

Parallel printout

Outputs the ASCII codes for 12 characters stored in the 6-word area specified by **s_Start** via the word output specified by **d** if the trigger **EN** is TRUE. If a printer is connected to the output specified by **d**, a character corresponding to the output ASCII code is printed.




## Parameters

Input

**s_Start** (WORD, INT, UINT)

starting 16-bit area for storing 12 bytes (6 words) of ASCII codes (source)

Output

**d** (WORD)

word output used for output of ASCII codes (destination)

## Remarks

- Only bit positions 0 to 8 of **d** are used in the actual printout. ASCII code is output in sequence starting with the lower byte of the starting area. Three scans are required for 1 character constant output. Therefore, 37 scans are required until all characters constants are output.
- Since it is not possible to execute multiple **F147_PR** instructions in one scan, use print-out flag **sys_bIsActive_F147_PR** to be sure they are not executed simultaneously.
  If the character constants convert to ASCII code, use of the **F95_ASC** instruction is recommended.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

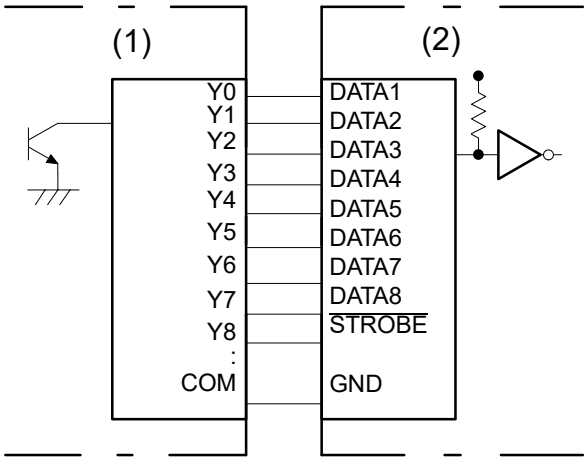
- if the ending area for storing ASCII codes exceeds the limit
- if the trigger of another **F147_PR** instruction turns to TRUE while one F147_PR instruction is being executed

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the ending area for storing ASCII codes exceeds the limit

- if the trigger of another **F147_PR** instruction turns to TRUE while one F147_PR instruction is being executed

## Connection example




(1) Transistor output type (output: 9 points or more)
(2) Printer (centronics interface)

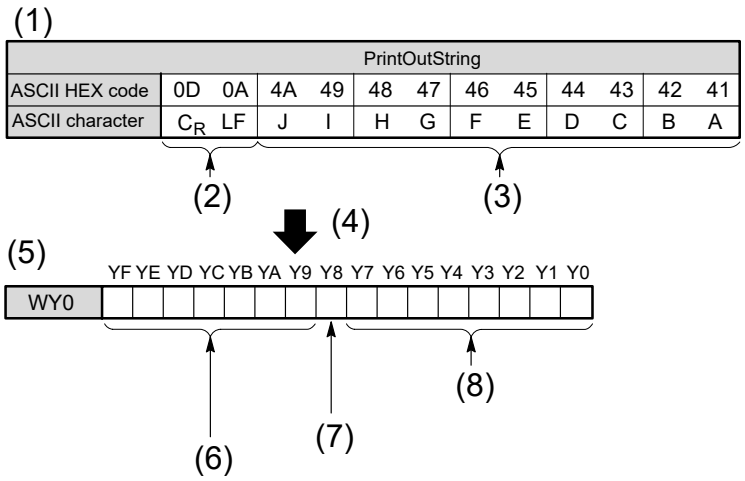
## Example

## Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global Variables

| | Class | Identifier | FP ... | IEC Address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | Printer | WY0 | %QW0 | WORD | 0 |
| 1 | VAR_GLOBAL | Print Out Flag | R9033 | %MX0.903.3 | BOOL | FALSE |



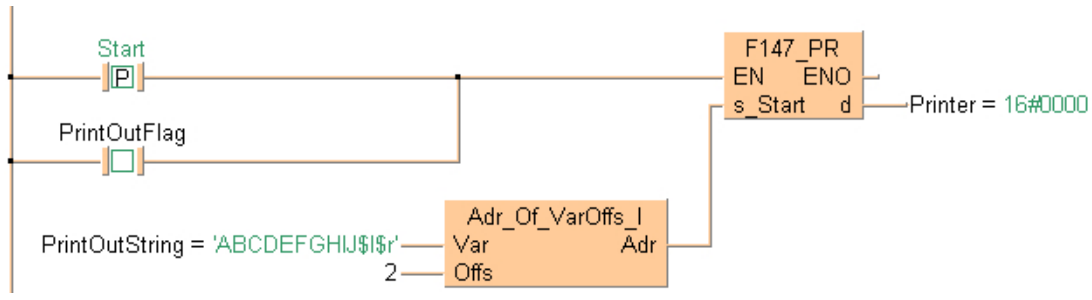

(1) Source: ASCII code for 12 characters A, B, C, D, E, F, G, H, I and J
(2) Control data for printer
(3) ASCII codes
(4) start: ON
(5) Destination
(6) Y9 to YF: not used
(7) Y8: for strobe signal of printer
(8) Y0 to YF: for data signals of printer (Y0 to Y7 correspond to DA A1 to DA A8 of printer)

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | Start | BOOL | FALSE | |
| 1 | VAR_EXTERNAL | PrintOutFlag | BOOL | FALSE | |
| 2 | VAR | PrintOutString | STRING[12] | 'ABCDEFGHIJ$L$R' | $L = line feed<br>$R = carriage return |
| 3 | VAR_EXTERNAL | Printer | WORD | 0 | |

## LD body

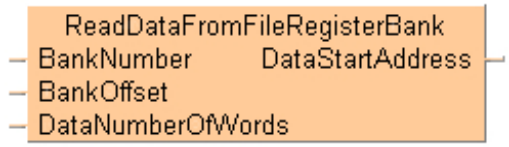


# 22 Memory device instructions

# ReadDataFromFileRegisterBank

Read data from file register bank 1 or 2

This instruction reads the number of words specified by **DataNumberOfWords** from File Register Bank 1 or 2, as specified by **BankNumber** beginning with **BankOffset**, and writes it to **DataStartAddress**.




## Parameters

Input

**BankNumber** (INT)

Specifies the bank number

**BankOffset** (INT)

Specifies the bank number offset

**DataNumberOfWords** (INT)

Number of word units to be read from the file register bank

Output

**DataStartAddress** (WORD, INT, UINT)

Specifies the start address of data which is read from the file register bank

## Remarks

With this function you cannot read data in the FL area (File Register Bank 0), i.e., the variable applied at **DataStartAddress** must not be located in the FL area.

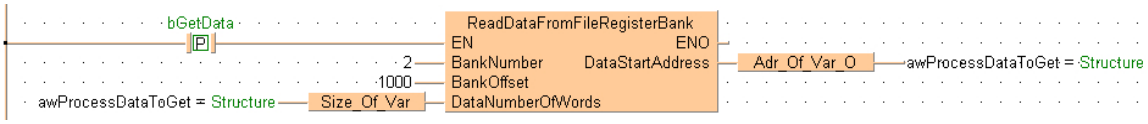
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | awProcessDataToStore | ARRAY [0..10] OF INT | [-111,111,222,333,444,555,666,777,888,999,1100] |
| 1 | VAR | awProcessDataToGet | ARRAY [0..10] OF INT | [11(0)] |
| 2 | VAR | bStoreData | BOOL | FALSE |
| 3 | VAR | bGetData | BOOL | FALSE |

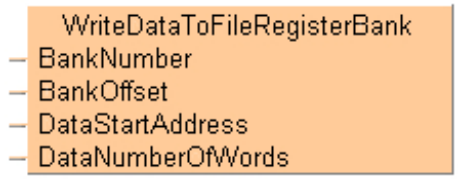
## LD body



# WriteDataToFileRegisterBank

Write data to file register bank 1 or 2

This instruction reads the number of words specified by **DataNumberOfWords** from **DataStartAddress** and writes it to the File Register Bank 1 or 2 as specified by **BankNumber** beginning with **BankOffset**.




## Parameters

Input

**BankNumber** (INT)

Specifies the bank number

**BankOffset** (INT)

Specifies the bank number offset

**DataStartAddress** (WORD, INT, UINT)

Specifies start address of data to be written to File Register Bank

**DataNumberOfWords** (INT)

Specifies number of word units to be written to File Register Bank

## Remarks

With this function you cannot write data to the FL area (File Register Bank 0), i.e., the variable applied at **DataStartAddress** must not be located in the FL area.

## Example

### POU header

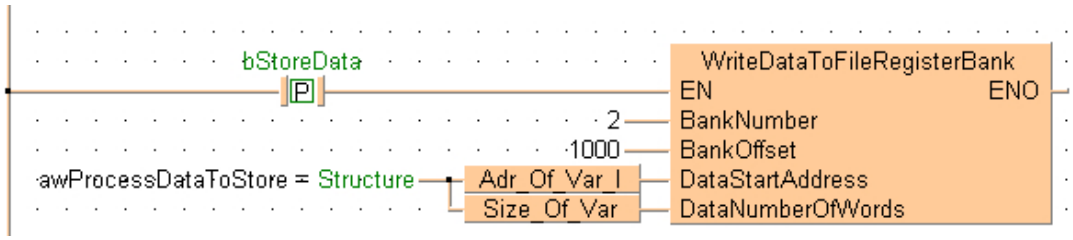
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | awProcessDataToStore | ARRAY [0..10] OF INT | [-111,111,222,333,444,555,666,777,888,999,1100] |
| 1 | VAR | awProcessDataToGet | ARRAY [0..10] OF INT | [11(0)] |
| 2 | VAR | bStoreData | BOOL | FALSE |
| 3 | VAR | bGetData | BOOL | FALSE |

## POU body

If **bStoreData** changes from FALSE to TRUE, the entire data unit variable **awProcessDataToStore** (a DUT containing 11 elements) is filled with the data from File Register Bank 2 BankOffset 1000.

## LD body



## 22.3 FP instructions

Tip

Advantages of FP instructions

### 22.3.1 Introduction to SD card instructions

#### Using SD card instructions with FP-XH Ethernet type PLCs

Tip

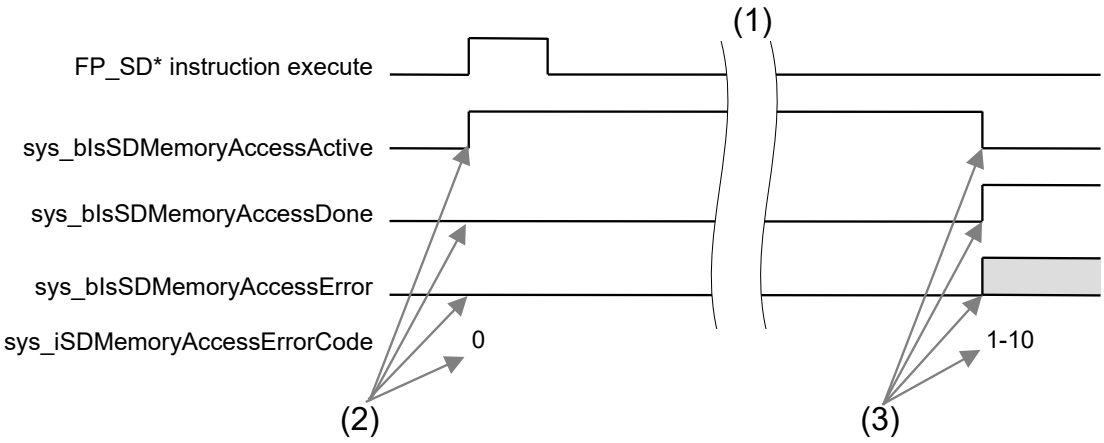
- You can use all SD card instructions with an FP-XH Ethernet PLC even though it has no SD card slot. Instead of using an SD card for reading/writing data, the data is stored internally in the RAM.
  Please note that data is not stored when the PLC is turned off.
- The memory size is limited to 1MB.
  **FP_SD_GET_FREE_KBYTES** returns the amount of free space.

#### Operation of instructions

- Upon execution of an SD card instruction, the following checks are conducted:
- Is an SD card present?
- Is the cover closed?
- Is the SD card write-protected?
- During the execution, the SD memory access active flag (**sys_bIsSDMemoryAccessActive**) is TRUE, and the SD memory access done flag (**sys_bIsSDMemoryAccessDone**) is FALSE.
- The execution of an SD card instruction is performed through multiple scans.
- Upon completion of the execution, the SD memory access error flag (**sys_bIsSDMemoryAccessError**) turns to TRUE or FALSE, depending on the result. Use this flag to judge whether the SD card instruction has completed normally or abnormally when the SD memory access done flag (**sys_bIsSDMemoryAccessDone**) turns to TRUE. The error code is stored in a system data register which can be accessed using **sys_iSDMemoryAccessErrorCode**.
- Only one type of SD card instruction can be executed at the same time. To execute more than one instruction, use flags such as the SD memory access active flag (**sys_bIsSDMemoryAccessActive**).

- Use **FP_SET_ERROR** to clear error flags.
- SD card instructions cannot be used in interrupt programs.

## Flag operation




(1) The execution of an SD card instruction is performed through multiple scans.

(2) All flags (SD memory access active, SD memory access done, SD memory access error) and registers (error code) are set upon execution of the instruction.

(3) Execution completion is announced at the end of the scan. If an error has occurred, the SD memory access error flag is set to TRUE and the error code is written into a system data register.

Note

When one of the following errors is detected, completion is announced immediately and the SD memory access active flag does not turn to TRUE:

- No SD card
- Write-protection of SD card enabled
- File/directory name error

## List of error codes

| Error code | Name of error | Cause | Affected instructions |
|---|---|---|---|
| 0 | Completed without error | | |
| 1 | No SD card | No SD card is installed, or the cover is open. | All SD card instructions at the time of execution. |

| Error code | Name of error | Cause | Affected instructions |
|---|---|---|---|
| 2 | Write-protection of SD card enabled | The SD card is write-protected. | • FP_SD_WRITE*<br>• FP_SD_DELETE*<br>• FP_SD_MOVE*<br>• FP_SD_COPY*<br>• FP_SD_RENAME* |
| 3 | File/directory name error | The syntax for specifying the file name and directory is wrong or too many subdirectories were specified. | • FP_SD_*_FILE<br>• FP_SD_*_DIR |
| 4 | File not specified | The specified file does not exist. | • FP_SD_*_FILE<br>• FP_SD_*_DIR |
| 5 | File exists | The specified file already exists. | • FP_SD_MOVE*<br>• FP_SD_COPY*<br>• FP_SD_RENAME* |
| 6 | Reading error | | All read instructions at the time of execution. |
| 7 | Writing error | The specified file is write-protected. | • FP_SD_WRITE*<br>• FP_SD_DELETE*<br>• FP_SD_MOVE*<br>• FP_SD_COPY*<br>• FP_SD_RENAME* |
| 8 | Wrong position | The reading or writing position is wrong. | • FP_SD_WRITE<br>• FP_SD_READ<br>• FP_SD_READ_LINE<br>At the time of execution. |
| 9 | SD card full | There is not enough free capacity on the SD card. | • FP_SD_WRITE*<br>• FP_SD_DELETE*<br>• FP_SD_MOVE*<br>• FP_SD_COPY*<br>• FP_SD_RENAME* |
| 10 | Wrong reading format | Error in the conversion format when reading a file. | • FP_SD_READ<br>At the time of execution. |
| 11 | File access conflict | The specified file is being logged or is being accessed via FTP. | • FP_SD_WRITE*<br>• FP_SD_DELETE*<br>• FP_SD_MOVE*<br>• FP_SD_COPY*<br>• FP_SD_RENAME* |
| -1 to -99 | Others | | All SD card instructions |

## How to specify directory and file names in an SD card

- Specify the full path (up to 256 characters).
- Do not specify the drive name.
- Specify a file extension, e.g. .txt.
- To save data in a file called abc.txt in the root directory, enter this: \abc.txt
- To save data in a file called def.txt in a subdirectory called "sub", enter this: \sub\def.txt
- To save data in a file called def.txt in a new subdirectory called "new", enter this: \new \def.txt

Note

- A subdirectory can only be created directly under its parent directory with one instruction. For directories in lower hierarchy levels, parent directories must be created first.
- If two or more files are specified, error 4 "File not specified" occurs.

## SD card specifications

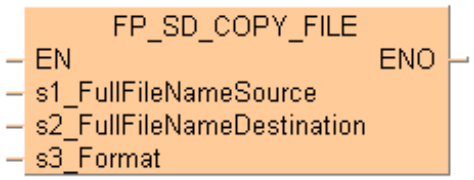
| | SD | SDHC |
|---|---|---|
| File system | FAT16 | FAT32 |
| Max. length of file name | Supports long file names (VFAT) | |
| Max. capacity | 2GB | 32GB |
| Max. file size | 2GB | 4GB |
| Max. number of files (8.3 format) in root directory | 512 | 65535 |
| Max. number of files (8.3 format) in sub directory | 65534 | 65534 |
| Max. number of files (long format) in root directory | 170 | 21845 |
| Max. number of files (long format) in sub directory | 56634 | 65534 |

| Item | Description |
|---|---|
| Long file name | 255 bytes (256 bytes including full path) |
| File name/directory name | ASCII characters (16#20–16#7E) |

# FP_SD_COPY_FILE

Copy file or directory on SD card

This FP instruction copies the file on the SD card specified by **s1_FullFileNameSource** to the file specified by **s2_FullFileNameDestination** according to the parameters specified by **s3_Format**.




## Parameters

Input

**s1_FullFileNameSource** (STRING)

Source file and directory

**s2_FullFileNameDestination** (STRING)

Destination file and directory

**s3_Format** (WORD)

File handling and format:
Bit 0:

- 0: Overwrite if file exists
- 1: Abnormal end if file exists

Bit 1–15: Reserved for the system

## Remarks

- Please also refer to Introduction to SD card instructions.
- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- If a directory is specified for **s1_FullFileNameSource** and a file for **s2_FullFileNameDestination**, error 3 "File/directory name error" occurs.
- If a directory is specified by **s1_FullFileNameSource**, all files in this directory are copied into the directory specified by **s2_FullFileNameDestination**. Subdirectories are not copied.

- Write-protected files will not be overwritten.
- If **s1_FullFileNameSource** and **s2_FullFileNameDestination** are identical, an error occurs regardless of the value of **s3_Format**.
- If a file is specified by **s1_FullFileNameSource** and a directory by **s2_FullFileNameDestination**, the file is copied into the directory.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

### POU header

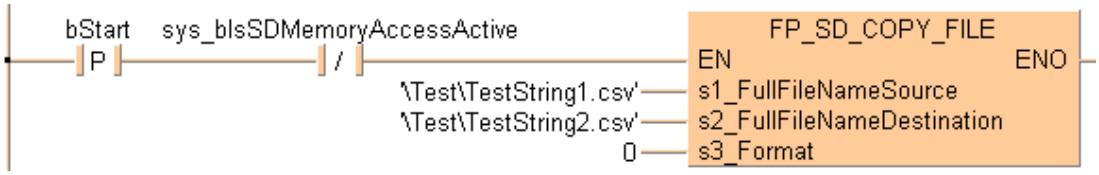
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |

### POU body

When the variable **bStart** changes from FALSE to TRUE and the system variable **sys_blsSDMemoryAccessActive** is not TRUE, the function is carried out. It copies the file '**TestString1.csv**' in the directory '**Test**' into the file '**TestString2.csv**' in the same directory.

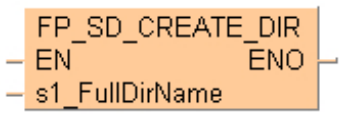
### LD body



# FP_SD_CREATE_DIR

Create directory on SD card

This FP instruction creates a directory on the SD card with the name specified by **s1_FullDirName**.




## Parameters

Input

**s1_FullDirName** (STRING)

Directory name: specifies the name of the directory.

## Remarks

- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- A subdirectory can only be created directly under its parent directory with one instruction. For directories in lower hierarchy levels, parent directories must be created first.
- If the directory to be created already exists, no error occurs.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |

## POU body

When the variable **bStart** changes from FALSE to TRUE and the system variable **sys_bIsSDMemoryAccessActive** is not TRUE, the function is carried out. It creates the directory '**Test**' in the root directory of the SD card.

## LD body

# FP_SD_DELETE_DIR

Delete directory on SD card

This FP instruction deletes the directory on the SD card specified by **s1_FullDirName**. If there are still files in the directory, use the instruction **FP_SD_DELETE_DIR_WITH_FILES**.


## Parameters

Input

**s1_FullDirName** (STRING)

Directory name: specifies the name of the directory.

## Remarks

- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- An error occurs when the directory to be deleted does not exist.
- An error occurs when the specified directory is not empty. Check the contents of the directory.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## POU header

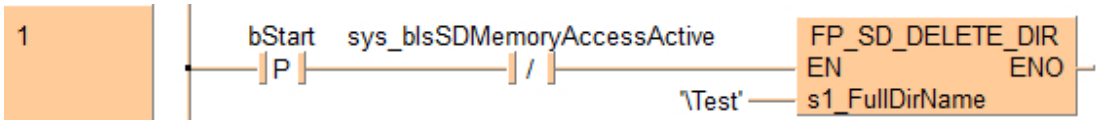
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |

## POU body

When the variable **bStart** changes from FALSE to TRUE and the system variable **sys_blsSDMemoryAccessActive** is not TRUE, the function is carried out. It deletes the folder '**Test**' from the root directory of the SD card.

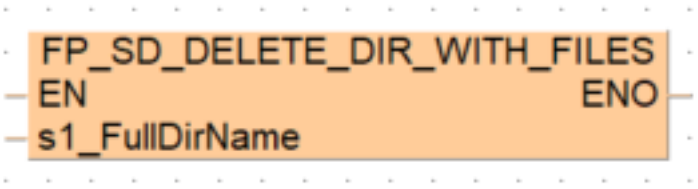
## LD body



# FP_SD_DELETE_DIR_WITH_FILES

Delete directory including files on SD card

This FP instruction deletes the directory including all files on the SD card specified by **s1_FullDirName**.




## Parameters

Input

**s1_FullDirName** (STRING[32])

Directory name: specifies the name of the directory.

## Remarks

- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- An error occurs when the directory to be deleted does not exist.
- An error occurs when the specified folder contains a subfolder.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

### POU header

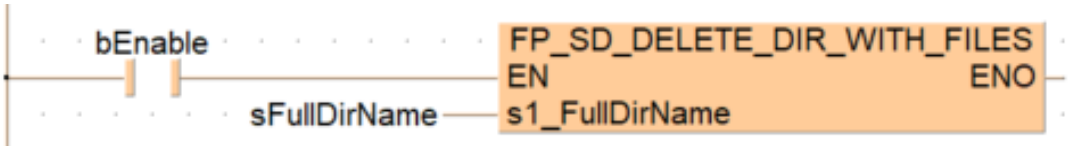
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | sFullDirName | STRING[32] | 'Logs' |
| 2 | VAR | bEnable | BOOL | FALSE |

### POU body

When the variable **bEnable** is set to TRUE, the function is executed. It deletes the folder '**Logs**' from the root directory including all files of the SD card.

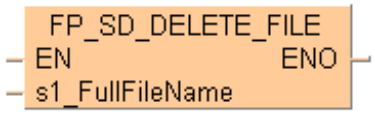
### LD body



# FP_SD_DELETE_FILE

Delete file from SD card

This FP instruction deletes the file specified by **s1_FullFileName** from the SD card.




## Parameters

Input

**s1_FullFileName** (STRING)

File name

## Remarks

- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- If the file to be deleted does not exist, an error occurs.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

## POU header

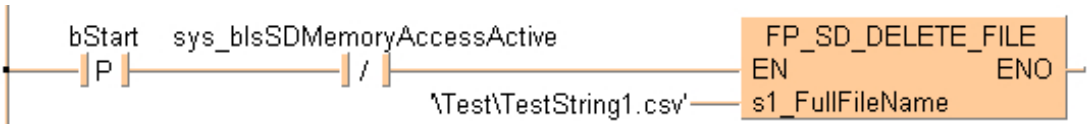
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |

### POU body

When the variable **bStart** changes from FALSE to TRUE and the system variable **sys_bIsSDMemoryAccessActive** is not TRUE, the function is carried out. It deletes the file 'TestString1.csv' from the directory 'Test' of the SD card.

### LD body



# FP_SD_GET_FILE_STATUS

Return the properties of a specified file on the SD card

This FP instruction returns the properties of the file on the SD card specified by **s_FullFileName** and stores the result in **d_dutStatus**.


## Parameters

Input

**s1_FullFileName** (STRING)

File name

Output

**d_dutStatus** (FP_SD_FILE_STATUS_DUT)

Contains the file properties

## Remarks

- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- The SD memory access done flag (**sys_bIsSDMemoryAccessDone**) is FALSE when the instruction is executed and turns to TRUE and remains TRUE when the instruction is completed.

## Error flags

**sys_bIsSDMemoryAccessError**

- FALSE: when the instruction has completed without error
- TRUE: when the instruction has completed with an error
- Use **sys_iSDMemoryAccessErrorCode** to evaluate the error code.

## Example

### POU header

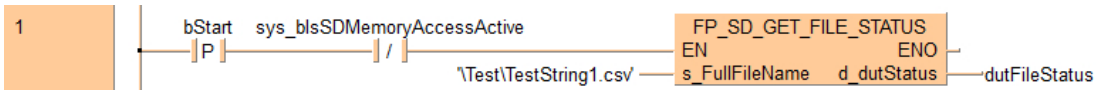
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |
| 1 | VAR | dutFileStatus | FP_SD_FILE_STATUS_DUT | | |

### POU body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out.

### LD body




| | | | | |
|---|---|---|---|---|
| 3 | dutFileStatus | | | |
| 4 | wFileAttribute | 16#0000 | DT19660 | File attribute Bit 0: 1 - read only file Bit 1: 1 - hidden file Bit 2: 1 - system |
| 5 | w1 | 16#0000 | DT19661 | Reserved |
| 6 | udiSize | 0 | DT19662 | File size |
| 7 | uiYear_2Digits | 0 | DT19664 | Year (0-99) |
| 8 | uiMonth | 0 | DT19665 | Month (1-12) |
| 9 | uiDay | 0 | DT19666 | Day (1-31) |
| 10 | uiHour | 0 | DT19667 | Hour (0-23) |
| 11 | uiMinute | 0 | DT19668 | Minute (0-59) |
| 12 | uiSecond | 0 | DT19669 | Second (0-59) |

# FP_SD_GET_FREE_KBYTES

Return the number of free kBytes on the SD card

This FP instruction calculates the free capacity in kilobytes available on the SD card and stores the result in **d_kBytes**.


## Parameters

Output

**d_kBytes** (UDINT)

Number of free kBytes

## Remarks

- Please also refer to Introduction to SD card instructions.
- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- The SD memory access done flag (**sys_bIsSDMemoryAccessDone**) is FALSE when the instruction is executed and turns to TRUE and remains TRUE when the instruction is completed.

## Error flags

**sys_bIsSDMemoryAccessError**

- FALSE: when the instruction has completed without error
- TRUE: when the instruction has completed with an error
  Use **sys_iSDMemoryAccessErrorCode** to evaluate the error code.

## Example

## POU header

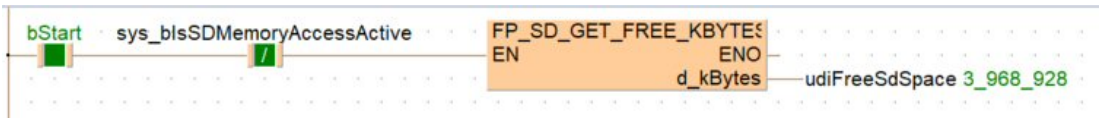
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instrution |
| 1 | VAR | udiFreeSdSpace | UDINT | 0 | |

## POU body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out. It calculates the number of free kBytes on the SD card and stores the value in **d_kBytes**.

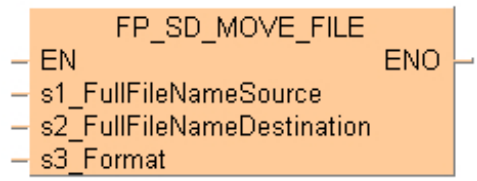
## LD body



# FP_SD_MOVE_FILE

Move file on SD card

This FP instruction moves the file specified by **s1_FullFileNameSource** to the file specified by **s2_FullFileNameDestination**.




## Parameters

Input

**s1_FullFileNameSource** (STRING)

Source file and directory

**s2_FullFileNameDestination** (STRING)

Destination file and directory

**s3_Format** (WORD)

File transfer format:
Bit 0:

- 0: Overwrite if file exists
- 1: Abnormal end if file exists

Bit 1–15: Reserved for the system

## Remarks

- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- If the directory specified by **s2_FullFileNameDestination** does not exist, error 3 "File/directory name error" occurs.
- If there is not enough free capacity, error 9 "SD card full" occurs.
- If a directory is specified for **s1_FullFileNameSource** and a file for **s2_FullFileNameDestination**, error 3 "File/directory name error" occurs.
- If a directory is specified by **s1_FullFileNameSource**, all files in this directory are moved to the directory specified by **s2_FullFileNameDestination**.

- Subdirectories are not moved.
- Write-protected files which are moved will keep their status.
- The free capacity on the SD card must be larger than the files to be moved.
- If a file is specified by **s1_FullFileNameSource** and a directory by **s2_FullFileNameDestination**, the file is moved to the directory.

### Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

### Example

### POU header

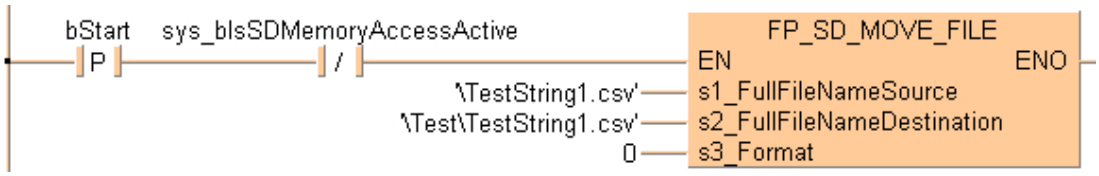
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |

### POU body

When the variable **bStart** changes from FALSE to TRUE and the system variable **sys_blsSDMemoryAccessActive** is not TRUE, the function is carried out. It moves the file '**TestString1.csv**' from the root directory of the SD card to the directory '**Test**'.

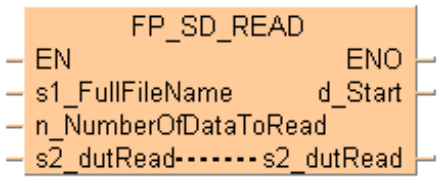
### LD body



# FP_SD_READ

Read data from SD card

This FP instruction reads the number of data specified by **n_NumberOfDataToRead** from a file specified by **s1_FullFileName** on the SD card. The DUT **FP_SD_READ_DUT** at **s2_dutRead** specifies the reading parameters. The result is stored in the area specified by **d_Start**. ASCII code requires 8 bits (1 byte) to express 1 BCD character. Upon conversion to ASCII, the data length will thus be twice the length of the source data.




## Parameters

Input

**s1_FullFileName** (STRING)

File name

**n_NumberOfDataToRead** (WORD, INT, UINT)

Number of data to be read
Values:

- 0–65535 (16-bit data)
- 0–32767 (32-bit data)
- 0–1999 (character string)

Input/output

**s2_dutRead** (FP_SD_READ_DUT)

Reading format, reading mode, pointer position, etc.

Output

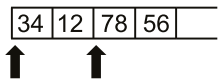
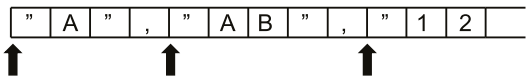
**d_Start** (ANY)

Starting address

## Pointer mode

If a pointer mode has been selected in FP_SD_READ_DUT at **s2_dutRead** (wMode=2 or wMode=3), reading is started at the position of the pointer. After reading, the pointer moves to the position where reading has been completed. The next reading operation will start from

this point.The position of the pointer can also be specified by udiBytePosition in the DUT. In the examples below, the arrows indicate valid pointer positions:

| 16-bit INT data (BIN format file): | ASCII data (CSV format file): |
|---|---|
|  |  |



In CSV files, the pointer always moves to the position next to a space, comma or line break. Spaces, commas and line breaks at the end of data are skipped.

## Remarks

- Please also refer to Introduction to SD card instructions.
- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- The SD memory access done flag (**sys_bIsSDMemoryAccessDone**) is FALSE when the instruction is executed and turns to TRUE and remains TRUE when the instruction is completed.
- Do not read the data from the output area until execution of the instruction is completed.
- Two successive double quotation marks ("") in character strings are read as one character ("). One double quotation mark by itself is ignored.
- In CSV files, null fields (e.g. parts with successive commas) are counted as part of the number of data to read, but the data is not stored.

## Error flags

**sys_bIsSDMemoryAccessError** (turns to TRUE and remains TRUE)

- FALSE: when the instruction has completed without error
- TRUE: when the instruction has completed with an error
- Use **sys_iSDMemoryAccessErrorCode** to evaluate the error code.

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

### POU header

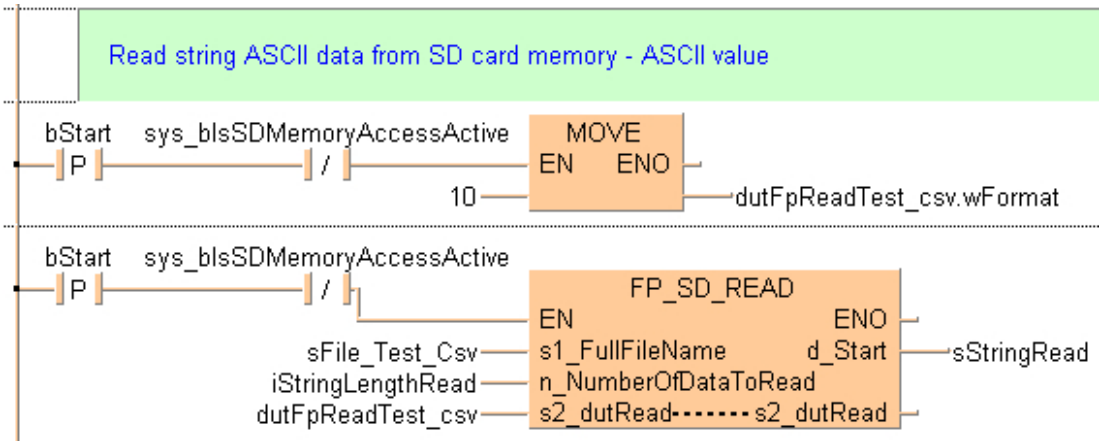
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |
| 1 | VAR | sFile_Test_Csv | STRING[32] | '\Test.csv' | |
| 2 | VAR | dutFpReadTest_csv | FP_SD_READ_DUT | | |
| 3 | VAR | sStringRead | STRING[32] | '' | |
| 4 | VAR | iStringLengthRead | INT | 0 | |

### POU body

When the variable **bStart** changes from FALSE to TRUE and the system variable **sys_blsSDMemoryAccessActive** is not TRUE, the function is carried out.

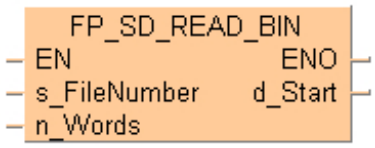
### LD body



# FP_SD_READ_BIN

Read binary data from SD card

This FP instruction reads the number of words specified by **n_Words** from the binary format file specified by the file number **s_FileNumber** on the SD card. The directory name is \data and the file name is dtxxx.bin where "xxx" is the file number. The result is stored in the area specified by **d_Start**. ASCII code requires 8 bits (1 byte) to express 1 BCD character. Upon conversion to ASCII, the data length will thus be twice the length of the source data.




## Parameters

Input

**s_FileNumber** (WORD, INT, UINT)

File number (0–999)

**n_Words** (WORD, INT, UINT)

Number of words (0–65535)

Output

**d_Start** (WORD, INT, UINT)

Starting address

## Remarks

- Please also refer to Introduction to SD card instructions.
- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- The SD memory access done flag (**sys_bIsSDMemoryAccessDone**) is FALSE when the instruction is executed and turns to TRUE and remains TRUE when the instruction is completed.
- Do not read the data from the output area until execution of the instruction is completed.
- If there is no directory \data or if there is no file with the specified file number in the directory, an error occurs.

- If the number of data in the file is less than the number of data to be read, data is read up to the number of data in the file.

## Error flags

**sys_bIsSDMemoryAccessError**

- FALSE: when the instruction has completed without error
- TRUE: when the instruction has completed with an error
- Use **sys_iSDMemoryAccessErrorCode** to evaluate the error code.

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

### DUT

With a Data Unit Type (DUT) you can define a data unit type that is composed of other data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

DUT_VALUES [DUT]

| | Identifier | Type | Initial |
|---|---|---|---|
| 0 | iValue | INT | 0 |
| 1 | uiValue | UINT | 0 |
| 2 | diValue | DINT | 0 |
| 3 | udiValue | UDINT | 0 |
| 4 | rValue | REAL | 0.0 |
| 5 | arrValue | ARRAY[0..15] of DWORD | [16(0)] |
| 6 | sStringValue | STRING[32] | '' |

### POU header

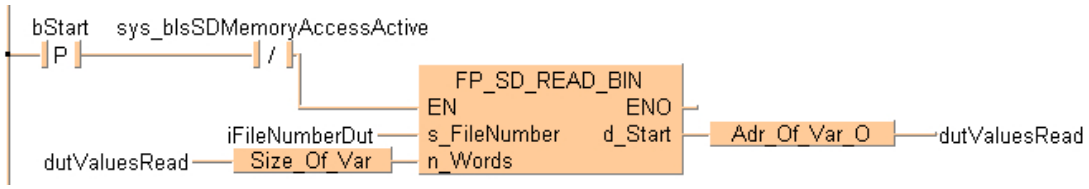
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |
| 1 | VAR | dutValuesRead | DUT_VALUES | | |
| 2 | VAR | iFileNumberDut | INT | 2 | |

### POU body

When the variable **bStart** changes from FALSE to TRUE and the system variable **sys_blsSDMemoryAccessActive** is not TRUE, the function is carried out.

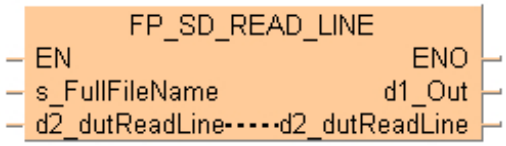
### LD body



# FP_SD_READ_LINE

Read one line from a specified file from SD card

This FP instruction reads the file specified by **s_FullFileName** from the pointer position specified by **d2_dutReadLine** up to a specified maximum number of bytes or until a line feed character (CR, LF, CR+LF) is detected. The result is stored in the area specified by **d1_Out**.




## Parameters

Input

**s_FullFileName** (STRING)

File name

Input/output

**d2_dutReadLine** (FP_SD_READ_LINE_DUT)

Pointer position and maximum number of characters to be read

Output

**d1_Out** (STRING)

Output area

## Remarks

- Please also refer to Introduction to SD card instructions.
- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- The SD memory access done flag (**sys_bIsSDMemoryAccessDone**) is FALSE when the instruction is executed and turns to TRUE and remains TRUE when the instruction is completed.
- Do not read the data from the output area until execution of the instruction is completed.
- Ensure that the number of bytes read (specified by **uiMaxCharsToRead** of **FP_SD_READ_LINE_DUT,**) does not exceed the maximum string length of the output variable at **d1_Out**. Otherwise, subsequent variables will be overwritten and unrelated operation errors can occur.

## Error flags

**sys_bIsSDMemoryAccessError**

- FALSE: when the instruction has completed without error
- TRUE: when the instruction has completed with an error
- Use **sys_iSDMemoryAccessErrorCode** to evaluate the error code.

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

### POU header

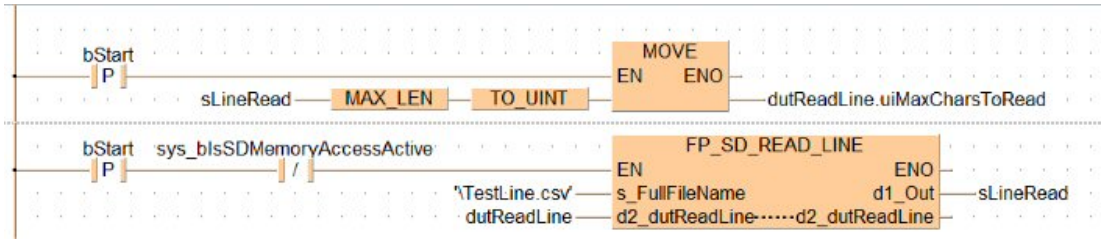
All input and output variables used for programming this function have been declared in the POU header.  The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bStart | BOOL | FALSE |
| 2 | VAR | dutReadLine | FP_SD_READ_LINE_DUT | |
| 3 | VAR | sLineRead | STRING[32] | '' |

### POU body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out.

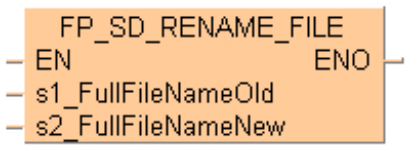
### LD body



# FP_SD_RENAME_FILE

Rename file on SD card

This FP instruction renames the file specified by **s1_FullFileNameOld** to the new file name specified by **s2_FullFileNameNew**.




## Parameters

Input

**s1_FullFileNameOld** (STRING)

Old file name

**s2_FullFileNameNew** (STRING)

New file name

## Remarks

- Please also refer to Introduction to SD card instructions.
- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- The SD memory access done flag (**sys_bIsSDMemoryAccessDone**) is FALSE when the instruction is executed and turns to TRUE and remains TRUE when the instruction is completed.

## Error flags

**sys_bIsSDMemoryAccessError**

- FALSE: when the instruction has completed without error
- TRUE: when the instruction has completed with an error
- Use **sys_iSDMemoryAccessErrorCode** to evaluate the error code.

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

### POU header

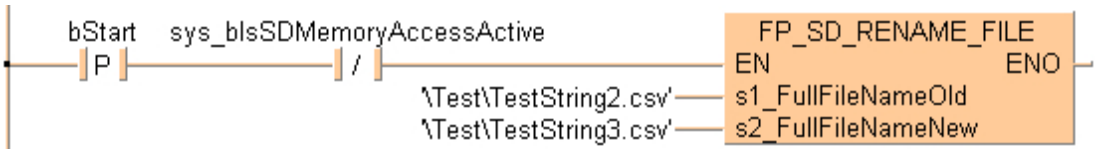
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |

### POU body

When the variable **bStart** changes from FALSE to TRUE and the system variable **sys_bIsSDMemoryAccessActive** is not TRUE, the function is carried out. It renames the file '**TestString2.csv**' in the directory '**Test**' into '**TestString3.csv**'.

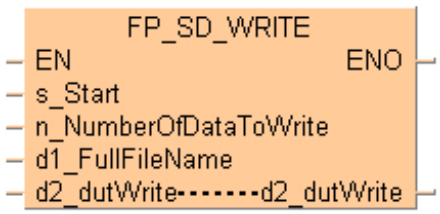
### LD body



# FP_SD_WRITE

Write data to SD card

This FP instruction reads the number of data specified by **n_NumberOfDataToWrite** stored at the address starting with **s_Start**, and writes the data to the file specified by **d1_FullFileName** on the SD card. The DUT **FP_SD_WRITE_DUT** at **d2_dutWrite** specifies the writing parameters.




## Parameters

Input

**s_Start** (ANY)

Starting address

**n_NumberOfDataToWrite** (WORD, INT, UINT)

Number of of data to write, specified by DUT member **udiNumberOfDataWritten**

- For **wFormat**=1–8: Number of 16-bit/32-bit comma-separated data
- For **wFormat**=10: Number of 8-bit characters
- For **wFormat**=11: Number of 16-bit data

**d1_FullFileName** (STRING)

File name with extension, e.g. '\Test.csv'

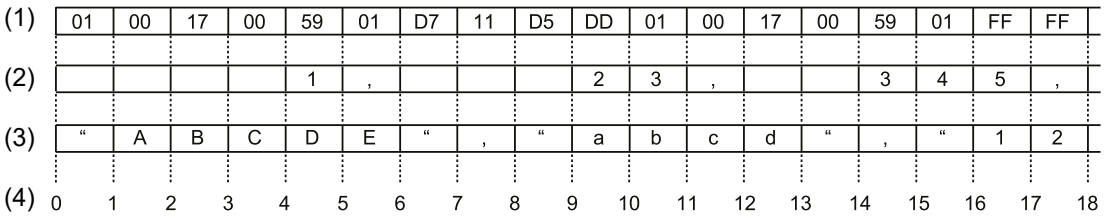
Input/output

**d2_dutWrite** (FP_SD_WRITE_DUT)

Writing format, writing mode, pointer position, etc.

## Pointer mode

If a pointer mode has been selected in **FP_SD_WRITE_DUT** at **s2_dutWrite** (wMode=2 or wMode=3), writing is started at the position of the pointer. After writing, the pointer moves to the position where writing has been completed. The next writing operation will start from this point.The position of the pointer can also be specified by udiBytePosition in the DUT. In the example below, file pointer positions are indicated for different data formats.




(1) 16-bit INT data (BIN format file)
(2) 16-bit INT data (CSV format file)
(3) ASCII data (CSV format file)
(4) File pointer

## Remarks

- Please also refer to Introduction to SD card instructions.
- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- The SD memory access done flag (**sys_bIsSDMemoryAccessDone**) is FALSE when the instruction is executed and turns to TRUE and remains TRUE when the instruction is completed.
- When the file attribute is read-only, no data can be written.
- When saving ASCII data, written character strings are enclosed in double quotation marks.
- A double quotation mark (") in character strings is converted to two double quotation marks ("").

## Error flags

**sys_bIsSDMemoryAccessError**

- FALSE: when the instruction has completed without error
- TRUE: when the instruction has completed with an error
- Use **sys_iSDMemoryAccessErrorCode** to evaluate the error code.

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

### POU header

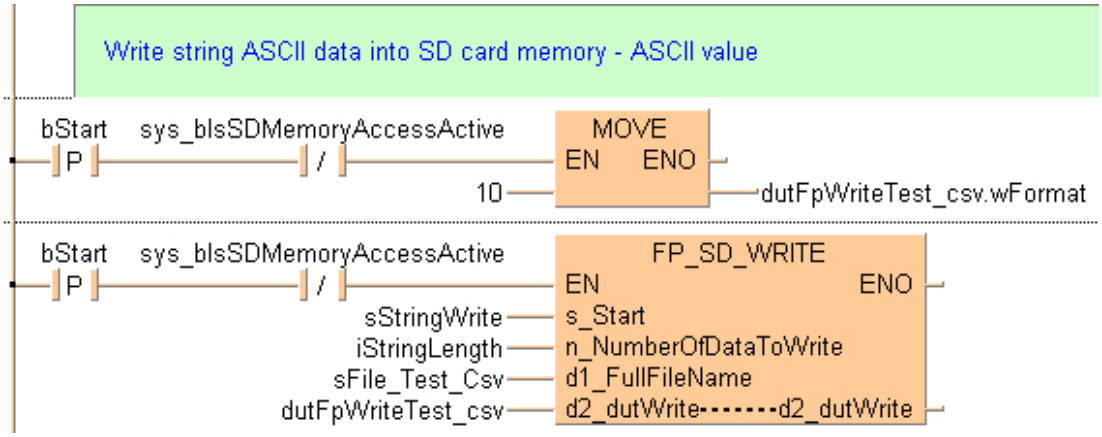
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |
| 1 | VAR | sFile_Test_Csv | STRING[32] | '\Test.csv' | |
| 2 | VAR | dutFpWriteTest_csv | FP_SD_WRITE_DUT | | |
| 3 | VAR | sStringWrite | STRING[32] | 'Test String!' | |
| 4 | VAR | iStringLength | INT | 0 | |

### POU body

When the variable **bStart** changes from FALSE to TRUE and the system variable **sys_bIsSDMemoryAccessActive** is not TRUE, the function is carried out.

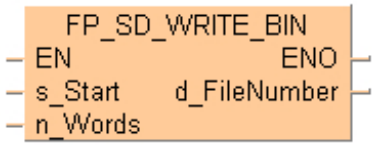
### LD body



# FP_SD_WRITE_BIN

Write binary data to SD card

This FP instruction reads binary data from a memory area specified by **s_Start** and **n_Words** and writes it into a binary format file specified by **d_FIleNumber** on the SD card. The directory name is \data, and the file name is dtxxx.bin where "xxx" is the file number.




## Parameters

Input

**s_Start** (WORD, INT, UINT)

Starting address

**n_Words** (WORD, INT, UINT)

Number of words

Output

**d_FileNumber** (WORD, INT, UINT)

File number (0–999)

## Remarks

- Please also refer to Introduction to SD card instructions.
- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- Make sure that **sys_bIsSDMemoryAccessDone** is FALSE.
- The SD memory access done flag (**sys_bIsSDMemoryAccessDone**) is FALSE when the instruction is executed and turns to TRUE and remains TRUE when the instruction is completed.
- If the specified folder does not exist, a new folder is created.
- If the file already exists, it will be overwritten.

### Error flags

**sys_bIsSDMemoryAccessError**

- FALSE: when the instruction has completed without error
- TRUE: when the instruction has completed with an error
- Use **sys_iSDMemoryAccessErrorCode** to evaluate the error code.

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

### Example

#### DUT

With a Data Unit Type (DUT) you can define a data unit type that is composed of other data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

DUT_VALUES [DUT]

| | Identifier | Type | Initial |
|---|---|---|---|
| 0 | iValue | INT | 0 |
| 1 | uiValue | UINT | 0 |
| 2 | diValue | DINT | 0 |
| 3 | udiValue | UDINT | 0 |
| 4 | rValue | REAL | 0.0 |
| 5 | arrValue | ARRAY[0..15] of DWORD | [16(0)] |
| 6 | sStringValue | STRING[32] | '' |

### POU header

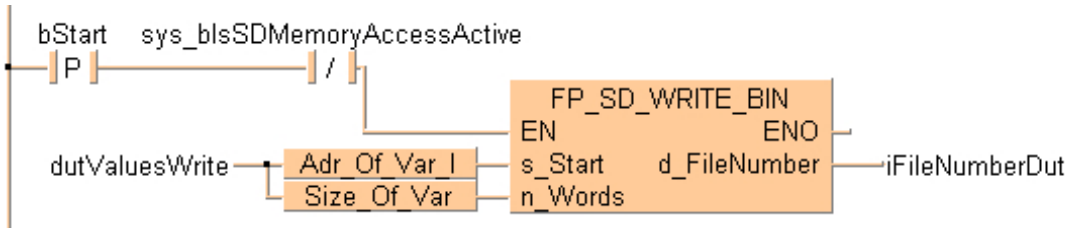
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |
| 1 | VAR | dutValuesWrite | DUT_VALUES | iValue := -32768,... | |
| 2 | VAR | iFileNumberDut | INT | 2 | |

### POU body

When the variable **bStart** changes from FALSE to TRUE and the system variable **sys_bIsSDMemoryAccessActive** is not TRUE, the function is carried out.

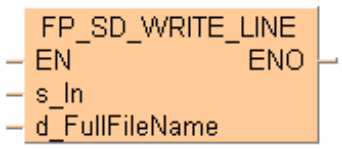
### LD body


bStart
P
sys_blsSDMemoryAccessActive
/
FP_SD_WRITE_BIN
EN
ENO
dutValuesWrite
Adr_Of_Var_I
Size_Of_Var
s_Start
n_Words
d_FileNumber
iFileNumberDut

# FP_SD_WRITE_LINE

Write one line to a specified file on SD card

This FP instruction writes the string specified by **s_In** into the file specified by **d_FullFileName** on the SD card.




## Parameters

Input

**s_In** (STRING)

String

**d_FullFileName** (STRING)

File name

## Remarks

- Please also refer to Introduction to SD card instructions.
- The SD memory access active flag (**sys_bIsSDMemoryAccessActive**) turns to TRUE after the trigger **EN** of the SD card instruction has turned to TRUE and remains TRUE until execution has completed. During this time, other SD card instructions cannot be executed.
- The SD memory access done flag (**sys_bIsSDMemoryAccessDone**) is FALSE when the instruction is executed and turns to TRUE and remains TRUE when the instruction is completed.
- If the specified file does not exist, a new file will be created.

## Error flags

**sys_bIsSDMemoryAccessError**

- FALSE: when the instruction has completed without error
- TRUE: when the instruction has completed with an error
- Use **sys_iSDMemoryAccessErrorCode** to evaluate the error code.

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

## Example

### POU header

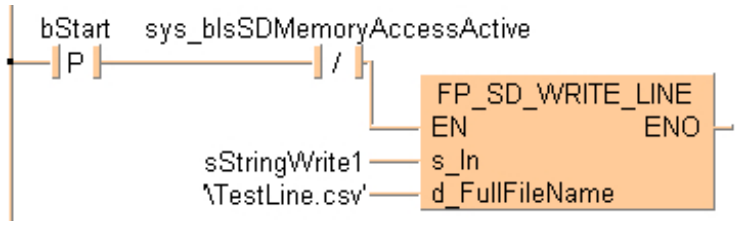
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |
| 1 | VAR | sStringWrite1 | STRING[32] | 'Test String1' | |

### POU body

When the variable **bStart** changes from FALSE to TRUE and the system variable **sys_bIsSDMemoryAccessActive** is not TRUE, the function is carried out.

### LD body




## Example writing multiple lines into an SD card

### POU header

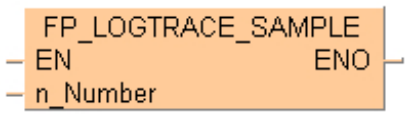
All input and output variables used for programming this function have been declared in the POU header.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 1 | VAR | i | INT | 0 | |
| 2 | VAR | asDumpString | ARRAY [0..99] OF STRING[32] | ['1','2','3','4','5','6','7','8',89(''),'z','y','z'] | |
| 3 | VAR | dutWriteInfo | FP_SD_WRITE_DUT | wFormat:=11,wMode:=1 | |
| 4 | VAR | sDumpString | STRING[3500] | '' | 100 * 32 character + separator |
| 5 | VAR | bDumpStringArrayToSDCard | BOOL | FALSE | |
| 6 | VAR | bStoreStringToSDCardFile | BOOL | FALSE | |

# FP_LOGTRACE_SAMPLE

Perform data recording triggered by instruction

This FP instruction creates a record with the data recording function (“Logging”/“Trace”) for the number specified by **n_Number**.




## Parameters

Input

**n_Number** (WORD, INT, UINT)

Data recording number 0–15
e.g. **SYS_LOG0**–**SYS_LOG15**

## Remarks

Make sure to set “Instruction (FP_LOGTRACE_SAMPLE)” for “Sampling trigger” in the “Data recording” window to use the data recording function with instruction otherwise an error message is output.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

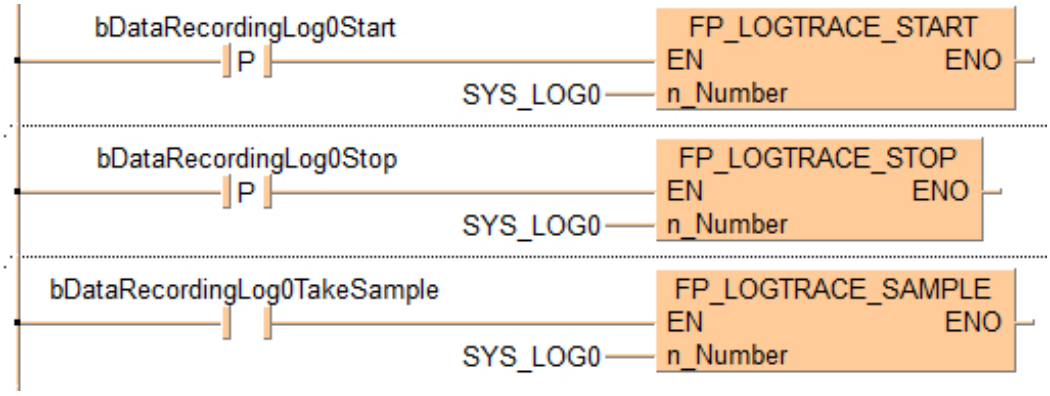
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bDataRecordingLog0Start | BOOL | FALSE |
| 1 | VAR | bDataRecordingLog0TakeSample | BOOL | FALSE |
| 2 | VAR | bDataRecordingLog0Stop | BOOL | FALSE |
| 3 | VAR | iDataRecordingValue1 | INT | 0 |
| 4 | VAR | iDataRecordingValue2 | INT | 0 |
| 5 | VAR | iDataRecordingValue3 | INT | 0 |
| 6 | VAR | iDataRecordingValue4 | INT | 0 |
| 7 | VAR | iDataRecordingValue5 | INT | 0 |

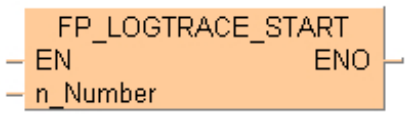
**LD body**



# FP_LOGTRACE_START

Start data recording

This FP instruction requests to start the data recording operation (“Logging”/“Trace”) for the number specified by **n_Number**.




## Parameters

Input

**n_Number** (WORD, INT, UINT)

Data recording number 0–15
e.g. **SYS_LOG0**–**SYS_LOG15**

## Remarks

- For starting the data recording operation, request the start after confirming that the data recording done flag (**sys_bIsLog0DataRecordingDone**, **sys_bIsLog1DataRecordingDone**, ...) has turned to TRUE.
- It takes a few milliseconds to a few seconds to start the data recording operation.
- There is no problem if a start request is made while the data recording operation is being started or in startup processing.
- Instead of using the instruction **FP_LOGTRACE_START**, you can also start the data recording operation automatically by checking “Start when changing to RUN mode” in the “Data recording” window.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a request is made to stop LOGn during startup processing
- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a request is made to stop LOGn during startup processing
- if the area specified using the index modifier exceeds the limit.

**sys_BIsLog0OperationError**–**sys_BIsLog15OperationError**

- if a request is made to stop LOGn during startup processing

- if the area specified using the index modifier exceeds the limit.

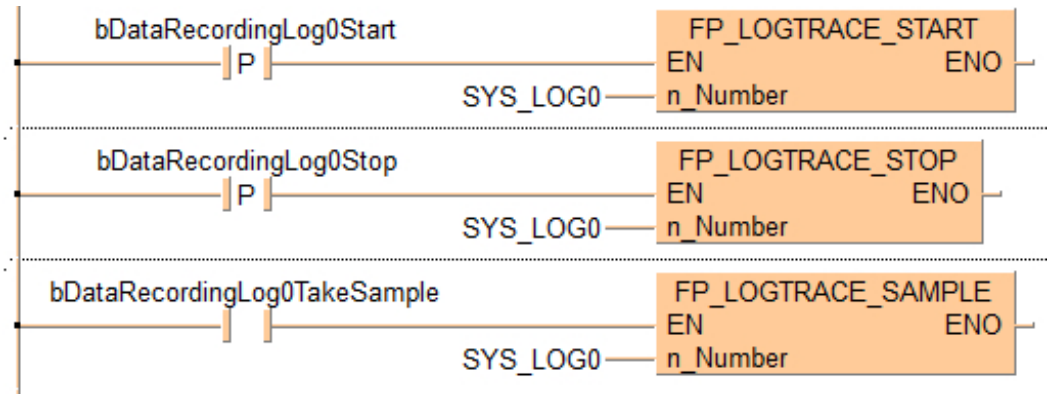
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bDataRecordingLog0Start | BOOL | FALSE |
| 1 | VAR | bDataRecordingLog0TakeSample | BOOL | FALSE |
| 2 | VAR | bDataRecordingLog0Stop | BOOL | FALSE |
| 3 | VAR | iDataRecordingValue1 | INT | 0 |
| 4 | VAR | iDataRecordingValue2 | INT | 0 |
| 5 | VAR | iDataRecordingValue3 | INT | 0 |
| 6 | VAR | iDataRecordingValue4 | INT | 0 |
| 7 | VAR | iDataRecordingValue5 | INT | 0 |

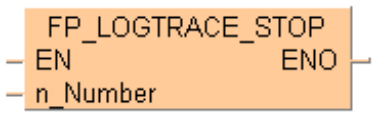
### LD body



# FP_LOGTRACE_STOP

Stop data recording

This FP instruction requests to stop the data recording operation (“Logging”/“Trace”) for the number specified by **n_Number**.




## Parameters

Input

**n_Number** (WORD, INT, UINT)

Data recording number 0–15
e.g. **SYS_LOG0**–**SYS_LOG15**

## Remarks

- For stopping the data recording operation, request the stop after confirming that the data recording active flag (**sys_bIsLog0DataRecordingActive**, **sys_bIsLog1DataRecordingDone**, ...) and SD memory access active flag (**sys_bIsLog0LoggingToSDCardActive**, **sys_bIsLog0LoggingToSDCardActive**,...) have turned to TRUE.
- It takes a few milliseconds to a few seconds to stop the data recording operation.
- There is no problem if a stop request is made while the data recording operation is being stopped or in stop processing.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a request is made to start LOGn during stop processing
- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a request is made to start LOGn during stop processing
- if the area specified using the index modifier exceeds the limit.

**sys_BIsLog0OperationError**–**sys_BIsLog15OperationError**

- if a request is made to start LOGn during stop processing
- if the area specified using the index modifier exceeds the limit.

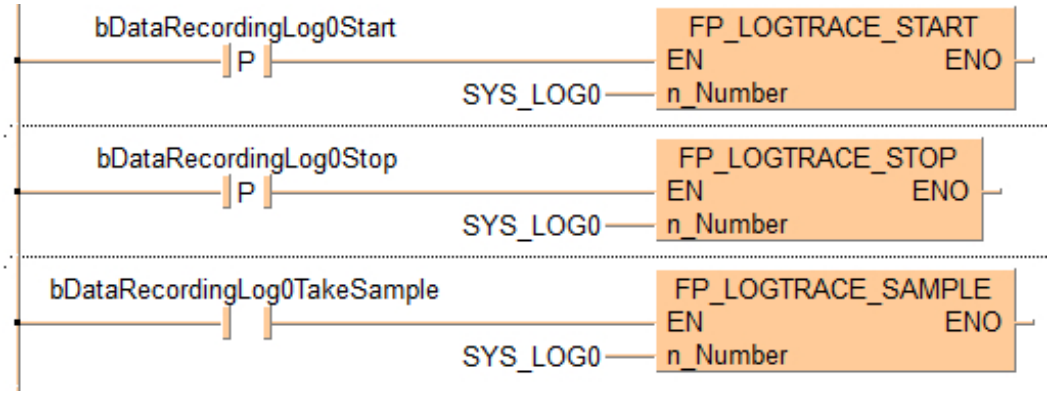
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bDataRecordingLog0Start | BOOL | FALSE |
| 1 | VAR | bDataRecordingLog0TakeSample | BOOL | FALSE |
| 2 | VAR | bDataRecordingLog0Stop | BOOL | FALSE |
| 3 | VAR | iDataRecordingValue1 | INT | 0 |
| 4 | VAR | iDataRecordingValue2 | INT | 0 |
| 5 | VAR | iDataRecordingValue3 | INT | 0 |
| 6 | VAR | iDataRecordingValue4 | INT | 0 |
| 7 | VAR | iDataRecordingValue5 | INT | 0 |

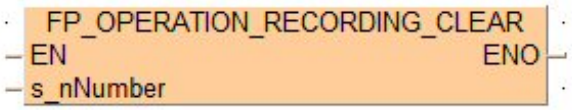
### LD body



# FP_OPERATION_RECORDING_CLEAR

Clear operation record

This instruction clears the operation record of the configuration number specified by **s_nNumber**.




## Parameters

Input

**s_nNumber** (ANY16)

Specifies the configuration number in the range from 0–7

## Remarks

Instead of using this instruction, you can also send the command from a web page created with Control Web Creator. Refer to the “Web Server Function Manual” for details.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if an unset configuration number has been specified.
- if the specified configuration number is outside the permissible range.
- if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if an unset configuration number has been specified.
- if the specified configuration number is outside the permissible range.
- if the area specified using the index modifier exceeds the limit.

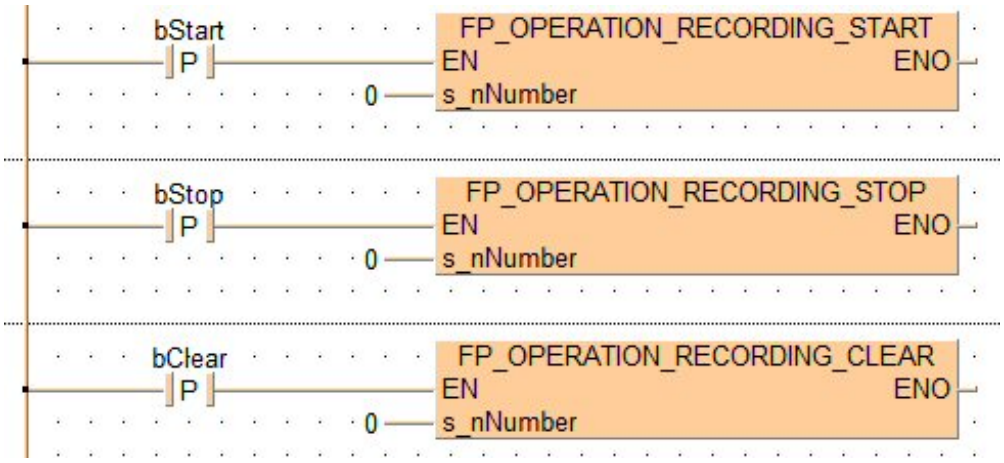
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bStart | BOOL | FALSE |
| 2 | VAR | bStop | BOOL | FALSE |
| 3 | VAR | bClear | BOOL | FALSE |

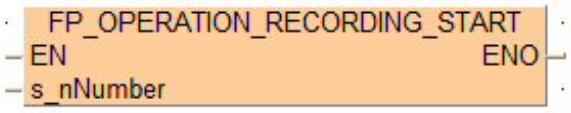
**LD body**



# FP_OPERATION_RECORDING_START

Start operation recording

This FP instruction requests to start the operation recording function (“Operation list” (“Alarms History”) and “Operation chart” (“Gantt Chart”) for the configuration number specified by **s_nNumber**.




## Parameters

Input

**s_nNumber** (ANY16)

Specifies the configuration number in the range from 0–7

## Remarks

Instead of using this instruction, you can also send the command from a web page created with Control Web Creator. Refer to the “Web Server Function Manual” for details.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if an unset configuration number has been specified.
- if the specified configuration number is outside the permissible range.
- if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if an unset configuration number has been specified.
- if the specified configuration number is outside the permissible range.
- if the area specified using the index modifier exceeds the limit.

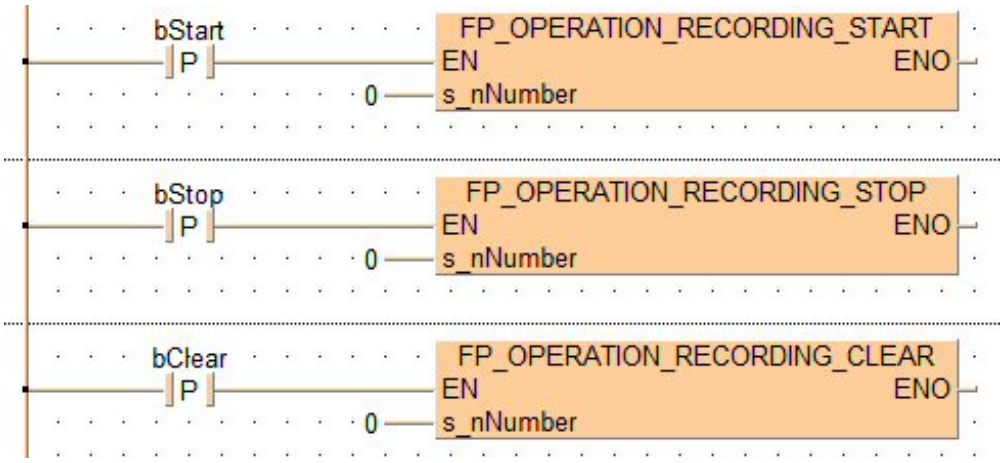
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bStart | BOOL | FALSE |
| 2 | VAR | bStop | BOOL | FALSE |
| 3 | VAR | bClear | BOOL | FALSE |

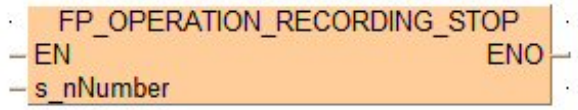
**LD body**



# FP_OPERATION_RECORDING_STOP

Stop operation recording

This FP instruction requests to stop the operation recording function (“Operation list” (“Alarms History”) and “Operation chart” (“Gantt Chart”) for the configuration number specified by **s_nNumber**.




## Parameters

Input

**s_nNumber** (ANY16)

Specifies the configuration number in the range from 0–7

## Remarks

Instead of using this instruction, you can also send the command from a web page created with Control Web Creator. Refer to the “Web Server Function Manual” for details.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if an unset configuration number has been specified.
- if the specified configuration number is outside the permissible range.
- if the area specified using the index modifier exceeds the limit.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if an unset configuration number has been specified.
- if the specified configuration number is outside the permissible range.
- if the area specified using the index modifier exceeds the limit.

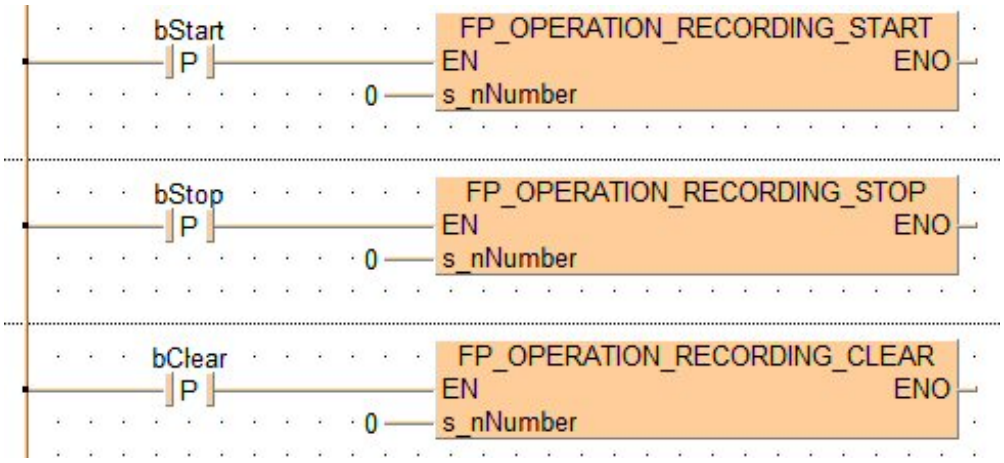
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bStart | BOOL | FALSE |
| 2 | VAR | bStop | BOOL | FALSE |
| 3 | VAR | bClear | BOOL | FALSE |

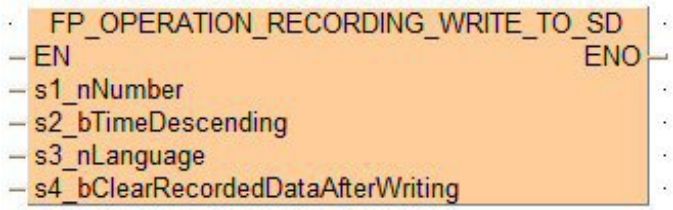
### LD body



# FP_OPERATION_RECORDING_WRITE_TO_SD

Write operation record to SD card

This FP instruction writes data recorded by the operation recording function of the configuration number specified by **s1_nNumber** to a CSV file, which is stored on the SD card in the OPH0–OPH7 folder (depending on the used configuration number). This folder is created automatically on the SD card.




## Parameters

Input

**s1_nNumber** (ANY16)

Specifies the configuration number in the range from 0–7

**s2_bTimeDescending** (BOOL)

Specifies whether the records should be written in ascending or descending order of their time stamp:

- FALSE: ascending
- TRUE: descending

**s3_nLanguage** (ANY16)

Specifies the language number defined in the “Operation recording” configuration:

- 0: language 0
- 1: language 1

**s4_bClearRecordedDataAfterWriting** (BOOL)

Specifies the behavior after writing to the SD card:

- FALSE: recorded data are not cleared
- TRUE: recorded data are cleared

## Remarks

- When this instruction is being executed, the system variable **sys_bIsSDMemoryAccessActive** is set to TRUE. When writing to SD card has finished, **sys_bIsSDMemoryAccessActive** is set to FALSE again.

- When this instruction is being executed, the system variable **sys_bIsSDMemoryAccessDone** is set to FALSE. When the recorded data has been written to the SD card, **sys_bIsSDMemoryAccessDone** is set to TRUE again.
- When this instruction has been successfully executed, the system variable **sys_bIsSDMemoryAccessError** is set to FALSE. When writing the recorded data to SD card was not successful, **sys_bIsSDMemoryAccessError** is set to TRUE.
- Instead of using this instruction, you can also send the command from a web page created with Control Web Creator. Refer to the “Web Server Function Manual” for details.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if an unset configuration number has been specified.
- if the specified configuration number is outside the permissible range.
- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if an unset configuration number has been specified.
- if the specified configuration number is outside the permissible range.
- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.

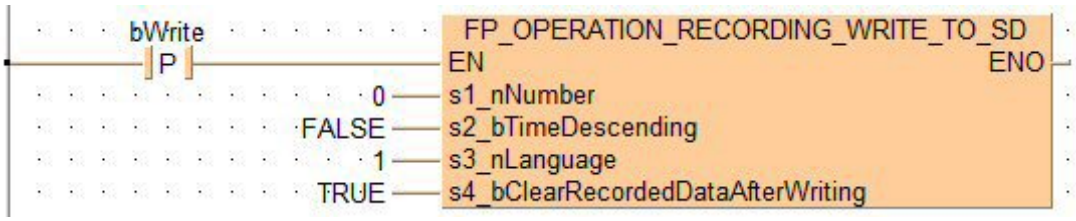
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bWrite | BOOL | FALSE |

## LD body



## 22.4 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

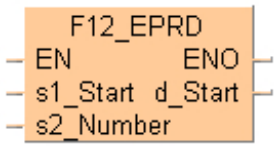
Advantages of IEC instructions

Advantages of FP instructions

# F12_EPRD

EEPROM read from memory

Using this instruction data will be copied from EEPROM/ Flash-ROM to the destination area (DT). The copy function is carried out with blocks only. Thus you can not copy single words. The block size and the number of blocks is shown in the table "PLC specific information". Also ensure that there at least 64/ 2048 free data registers (1 block = 64 words/ 2048 words (DTs)) reserved for the destination area.




## Parameters

Input

**EN** (BOOL)

Activation of the function (when **EN** has the state TRUE, the function block will be executed at every PLC scan)

**s1_Start** (DWORD, DINT, UDINT, DATE, TOD, DT)

EEPROM start block number

**s2_Number** (DWORD, DINT, UDINT, DATE, TOD, DT)

Number of blocks to be read. The block size is 64 or 2048 data registers depending on the PLC type (see the following table "PLC specific information").

Output

**ENO** (BOOL)

When the function was executed, ENO is set to TRUE. Helpful at cascading functions with EN-functionality

**d_Start** (WORD, INT, UINT)

DT start address for information to be written

### PLC specific information

| PLC type | FP0 2,7k C10/C14/ C1, FP-e | FP0 5k C32 | FP0 10k T32CP | FP-Sigma, FP-X, FP0R | FP0H |
|---|---|---|---|---|---|
| ROM | EEPROM | EEPROM | EEPROM | Flash-ROM | Flash-ROM |
| Block size (1 block) | 64 words (64x16bit) | 64 words (64x16bit) | 64 words (64x16bit) | 2048 words (2048x16bit) | 2048 words (2048x16bit) |
| EEPROM start block number | 0 to 9 | 0 to 95 | 0 to 255 | 0 to 15 | 0 to 31 |
| Number of blocks to be read / written each execution | 1 to 2 | 1 to 8 | 1 to 255 | 1 (writing) 1 to 16 (reading) | 1 (writing) 1 to 32 (reading) |
| Write duration (additional scan time) | < 20 ms each block | < 5 ms each block | < 5 ms each block | < 100ms each block | < 100ms each block |
| Read duration (additional scan time) | Less than 1 ms each block | Less than 1 ms each block | Less than 1 ms each block | 9.94μs + (1562.6*number of blocks) μs | |
| Max number of writing events | 100,000 | 10,000 | 10,000 | 10,000 | 10,000 |
| Max read times | No limit | No limit | No limit | No limit | No limit |

Note

Power down, RUN -> PROG mode changes are also counted.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

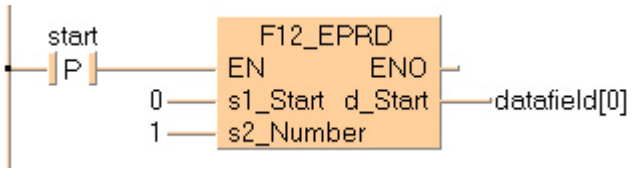
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the instruction |
| 1 | VAR | datafield | ARRAY [0..63] OF INT | [64(0)] | data field to be uploaded data from EEPROM |

### POU body

When the variable **start** changes from FALSE to TRUE, the function is carried out.

The function reads the first block (= 64 words) after start block number 0 from the EEPROM and writes the information into the data fields from **datafield[0]** until **datafield[63]**.

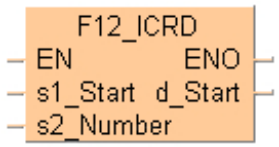
**LD body**


start
P
F12_EPRD
EN
ENO
0
s1_Start
d_Start
datafield[0]
1
s2_Number

# F12_ICRD

IC card extended memory read

The data for the number of words specified by **s2_Number** are read from the address in the IC card extended memory area specified by **s1_Start** and written to the area specified by **d_Start**.




## Parameters

Input

**s1_Start** (DWORD, DINT, UDINT, DATE, TOD, DT)

starting 32-bit area to be read in extended memory

**s2_Number** (DWORD, DINT, UDINT, DATE, TOD, DT)

number of words to be read

Output

**d_Start** (WORD, INT, UINT)

destination, starting 16-bit area

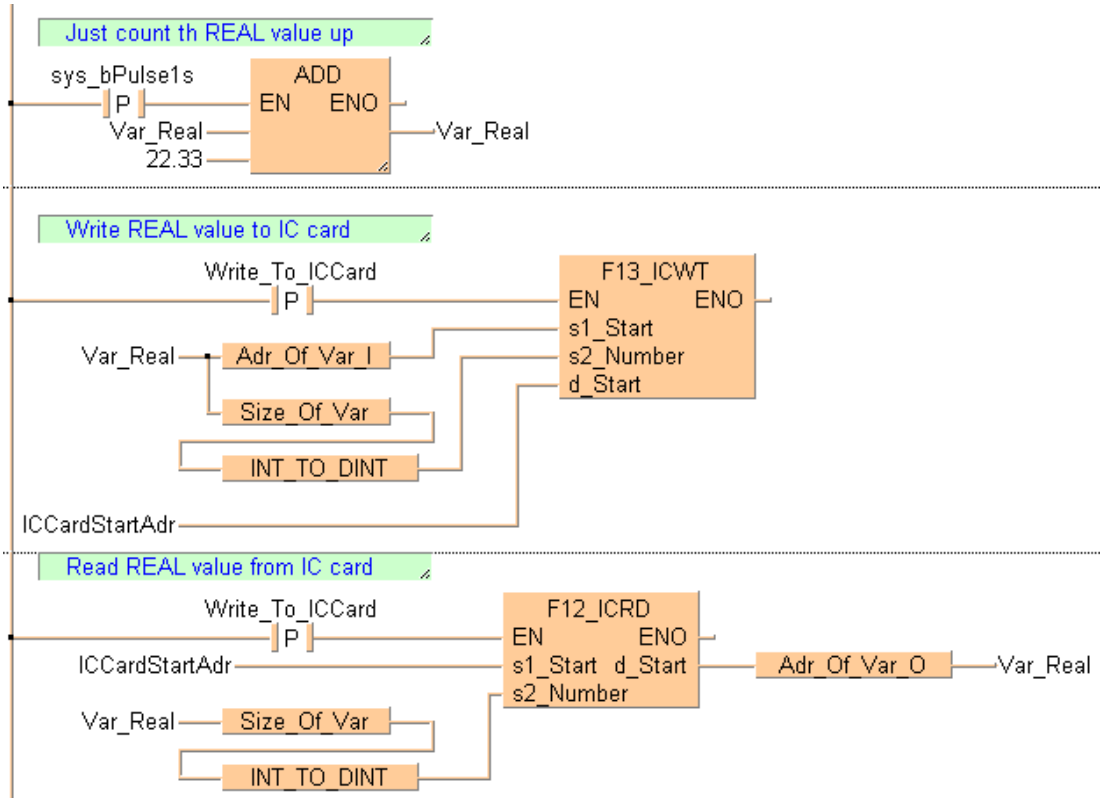
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Var_Real | REAL | 0.0 |
| 1 | VAR | Write_To_ICCard | BOOL | FALSE |
| 2 | VAR | Read_From_ICCard | BOOL | FALSE |
| 3 | VAR | ICCardStartAdr | DINT | 0 |

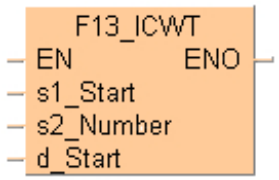
### LD body


Just count th REAL value up
sys_bPulse1s
P
ADD
EN
ENO
Var_Real
22.33
Var_Real
Write REAL value to IC card
Write_To_ICCard
P
F13_ICWT
EN
ENO
s1_Start
s2_Number
d_Start
Var_Real
Adr_Of_Var_I
Size_Of_Var
INT_TO_DINT
ICCardStartAdr
Read REAL value from IC card
Write_To_ICCard
P
F12_ICRD
EN
ENO
s1_Start
d_Start
s2_Number
ICCardStartAdr
Adr_Of_Var_O
Var_Real
Var_Real
Size_Of_Var
INT_TO_DINT

# F13_ICWT

IC card extended memory write

The data for the number of words specified by **s2_Number** are read from the address specified by **s1_Start** and written to the extended memory area in the IC card specified by **d_Start**.




## Parameters

Input

**s1_Start** (WORD, INT, UINT)

source data, starting 16-bit area

**s2_Number** (DWORD, DINT, UDINT, DATE, TOD, DT)

number of words to be read then written to IC card

**d_Start** (DWORD, DINT, UDINT, DATE, TOD, DT)

destination area of IC card expansion memory

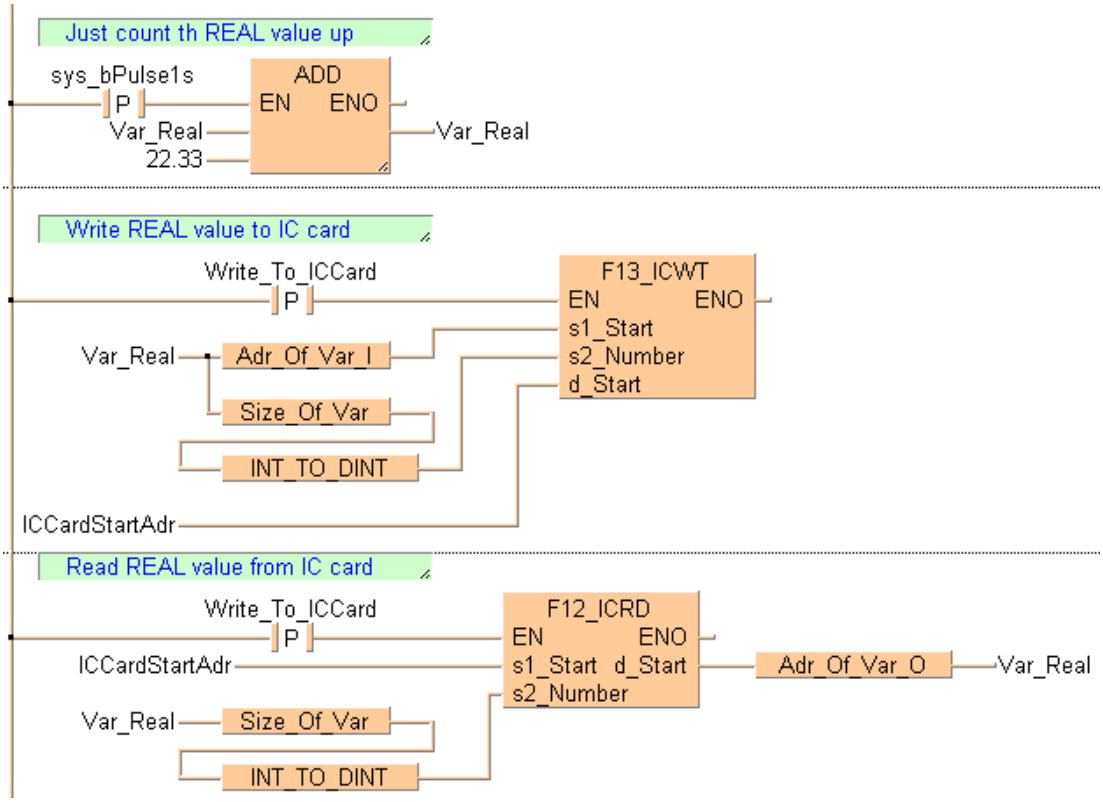
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Var_Real | REAL | 0.0 |
| 1 | VAR | Write_To_ICCard | BOOL | FALSE |
| 2 | VAR | Read_From_ICCard | BOOL | FALSE |
| 3 | VAR | ICCardStartAdr | DINT | 0 |

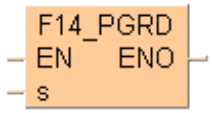
## LD body


Just count th REAL value up
sys_bPulse1s
P
ADD
EN
ENO
Var_Real
22.33
Var_Real
Write REAL value to IC card
Write_To_ICCard
P
F13_ICWT
EN
ENO
s1_Start
s2_Number
d_Start
Var_Real
Adr_Of_Var_I
Size_Of_Var
INT_TO_DINT
ICCardStartAdr
Read REAL value from IC card
Write_To_ICCard
P
F12_ICRD
EN
ENO
s1_Start
d_Start
s2_Number
ICCardStartAdr
Adr_Of_Var_O
Var_Real
Var_Real
Size_Of_Var
INT_TO_DINT

# F14_PGRD

Program read from IC card

When the execution criterion of **F14_PGRD** is turned ON, the execution proceeds until the END. The program subsequently switches to the program specified by **s**.




## Parameters

Input

**s** (ARRAY [0..5] OF WORD)

starting address of area storing program

## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

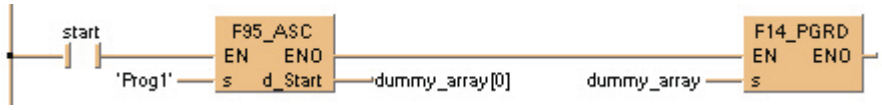
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | dummy_array | ARRAY [0..5] OF WORD | [6(0)] | contains the file name in HEX_ASCII format |
| 2 | VAR | | | | |

## POU body

When the variable **start** is set to TRUE, the function is carried out.

The instruction reads the program Prog1 from the IC card and executes it.

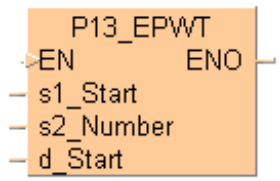
## LD body



# P13_EPWT

EEPROM write to memory

Using this instruction data will be copied from the data area (DT) to the EEPROM/ Flash-ROM.




## Parameters

Input

**EN** (BOOL)

Activation of the function (when **EN** changes from FALSE to TRUE, the function will be executed one time)

**s1_Start** (WORD, INT, UINT)

DT start address of the block(s) that you want to save

**s2_Number** (DWORD, DINT, UDINT, DATE, TOD, DT)

Number of blocks to be written. The block size is 64 or 2048 data registers depending on the PLC type (see the following table "PLC specific information").

**d_Start** (DWORD, DINT, UDINT, DATE, TOD, DT)

EEPROM start block number

Output

**ENO** (BOOL)

When the function was executed, ENO is set to TRUE. Helpful at cascading functions with EN-functionality

## Remarks

The EEPROM memory is not the same as the hold area. The hold area stores data in real time. Whenever the power shuts down, the hold data is stored in the EEPROM memory. The **P13_EPWT** instruction sends data into the EEPROM only when the instruction is executed. It also has a limitation of the number of times you can write to it (see table below). You must make sure that the **P13_EPWT** instruction will not be executed more often than the specified number of writes.

For example, if you execute **P13_EPWT** with R901A flag (pulse time 0.1s), the EEPROM will become inoperable after 100,000 * 0.1 sec=10,000 sec (2.8 hours). However if you want to hold your profile data such as positioning parameters or any other parameter values that are changed infrequently, you will find this instruction very useful.

Note

One of the two input variables **s2_Number** or **d_Start** has to be assigned constant number value.

### PLC specific information

| PLC type | FP0 2,7k C10/C14/ C1, FP-e | FP0 5k C32 | FP0 10k T32CP | FP-Sigma, FP-X, FP0R | FP0H |
|---|---|---|---|---|---|
| ROM | EEPROM | EEPROM | EEPROM | Flash-ROM | Flash-ROM |
| Block size (1 block) | 64 words (64x16bit) | 64 words (64x16bit) | 64 words (64x16bit) | 2048 words (2048x16bit) | 2048 words (2048x16bit) |
| EEPROM start block number | 0 to 9 | 0 to 95 | 0 to 255 | 0 to 15 | 0 to 31 |
| Number of blocks to be read / written each execution | 1 to 2 | 1 to 8 | 1 to 255 | 1 (writing)<br>1 to 16 (reading) | 1 (writing)<br>1 to 32 (reading) |
| Write duration (additional scan time) | < 20 ms each block | < 5 ms each block | < 5 ms each block | < 100ms each block | < 100ms each block |
| Read duration (additional scan time) | Less than 1 ms each block | Less than 1 ms each block | Less than 1 ms each block | 9.94μs + (1562.6*number of blocks) μs | |
| Max number of writing events | 100,000 | 10,000 | 10,000 | 10,000 | 10,000 |
| Max read times | No limit | No limit | No limit | No limit | No limit |

Note

Power down, RUN -> PROG mode changes are also counted.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

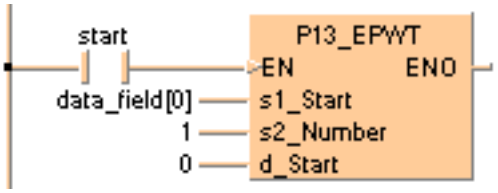
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |
| 1 | VAR | DataField | ARRAY (0..63] OF INT | (1,2,3,4,5,6,7,8,9,10,11,12,52(0)] | datafield to be uploaded data from EEPROM |

## POU body

When the variable **start** changes from FALSE to TRUE, the function is carried out.

The function reads the contents of data field[0] until data field[63] (s2* = 1 => 1 block = 64 words) and writes the information after start block number 0 into the EEPROM.

## LD body

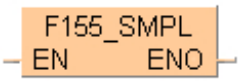


# F155_SMPL

Transfer sampling data

This instruction transfers the sampling data specified by the sampling trace editor into the sampling memory.

**F155_SMPL** can only be used with the sampling mode "per Scan".




## Remarks

Specify the sampling points using `Control FPWIN Pro`:

- Flags: 16 points
  Available for (FP format): X, Y, R, L, T, C
- Data: 3 words
  Available for (FP format): WX, WY, WR, WL, SV, EV, DT, LD, FL

## Example

### POU header

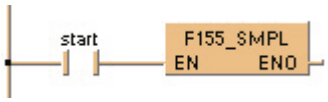
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

### LD body

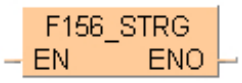


# F156_STRG

Set sampling trigger

This instruction sets the sampling trigger that stops the sampling after the delay specified by the sampling trace parameters.

**F156_STRG** can be used with both sampling modes "per Scan" and "per Time Interval".




## Remarks

Specify the sampling points using Control FPWIN Pro:

- Flags: 16 points
  Available for (FP format): X, Y, R, L, T, C
- Data: 3 words
  Available for (FP format): WX, WY, WR, WL, SV, EV, DT, LD, FL

## Example

### POU header

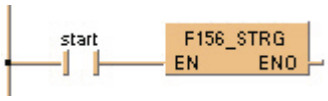
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

### LD body

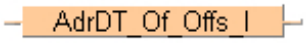


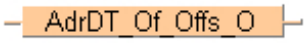
# 23 Pointer instructions

# AdrDT_Of_Offs

Returns the DT address of the input or output with offset

From the value at input/output **Offs**, the function generates a 16-bit address in the DT area at input/output **AdrDT**. This input has to be directly attached to a 16-bit input/output of a basic function. The value at input/output **Offs** represents the address offset within the DT data area. Using **GetPointer** it can be transferred along with a value for the memory area to a user function or function block.




## Parameters

Input

**AdrDT** (INT)

Yields 16-bit DT offset address

This pin must be connected to the input/output of a basic function for which the data type INT, WORD is allowed

Output

**Offs** (WORD, INT, UINT)

Offset for address DT0

## Remarks

- Remember, the execution of the basic function that uses the address created in the DT area is determined by assigning the function **Is_AreaDT** with the value for the memory area of the variable at EN-input.
- This function internally accesses one of the index registers that is also used for array calculations.
- Only for LD and FBD Editors: Use “Input instruction” or “Output instruction” from the “Instructions” pane to insert the instruction required into the programming window.

## Example

### DUT

DUT_NonBoolean [DUT]

| | Identifier | Type | Initial |
|---|---|---|---|
| 0 | Int1 | INT | 0 |
| 1 | ArrayOfInt1 | ARRAY [0..2] OF INT | [3(0)] |
| 2 | Word1 | WORD | 0 |
| 3 | ArrayOfDint1 | ARRAY [0..2] OF DINT | [3(0)] |
| 4 | Dword1 | DWORD | 0 |
| 5 | Dint1 | DINT | 0 |

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

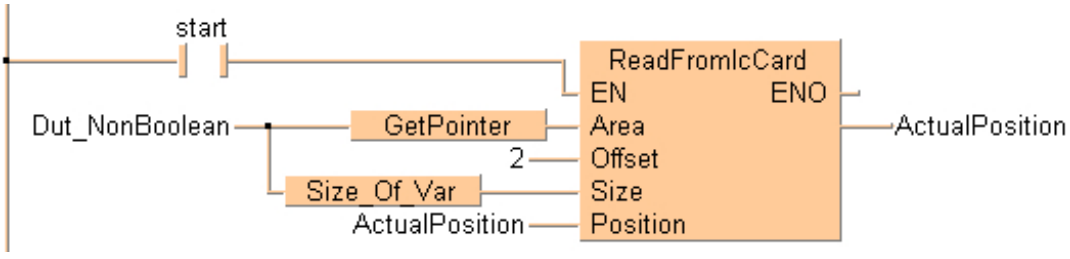
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activation of the function |
| 1 | VAR | DutNonBoolean | DUT_NonBoolean | | structured data type |
| 2 | VAR | Actual Position | DINT | 0 | Beginning position from which the data should be read to the IC card |

Here the variable **DutNonBoolean** of the data type assigned in the above DUT is declared. Assigning elements of the variable **DutNonBoolean** with values was not done in the POU header or body, since the values of the variable **DutNonBoolean** are overwritten after the function **ReadFromIcCard** is executed.

### POU body

When the variable **start** changes from FALSE to TRUE, the function **ReadFromIcCard** is carried out. The function reads values on the IC card beginning with address **ActualPosition** and writes the information to the variable **DutNonBoolean**. Do not forget that the IC card has to be appropriately formatted via the menu “Online” > “IC memory card manager...” and that, if necessary, values beginning at address **ActualPosition** should be present on the IC card.

### LD body



## ReadFromIcCardExample: programming the user-defined function :

### POU header

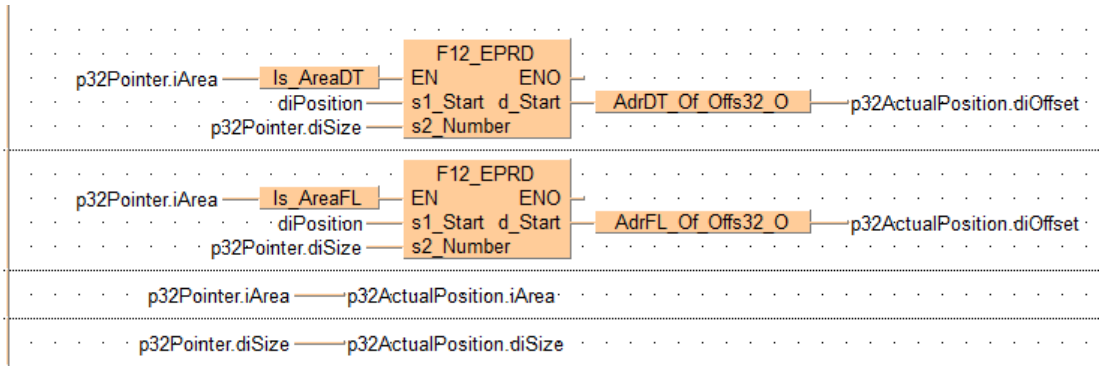
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_INPUT | p32Pointer | POINTER32 | | area, offset and size of data values |
| 1 | VAR_INPUT | diPosition | DINT | 0 | beginning position from which the data should be read from the IC card |
| 2 | VAR_OUTPUT | p32ActualPosition | POINTER32 | | shows the area, offset and size where the data of the IC card is stored |

### POU body

If the variable whose values for **Area** and **Offset** are delivered via the program called up lies in the DT- or FL-area (because the address for the variable **p32Pointer** is assigned to the DT- or FL-area by the compiler), the data is read from the IC card beginning with **p32ActualPosition**. The position is raised by the size of this variable and returned.

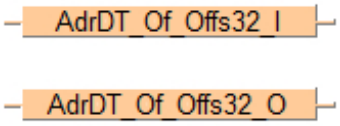
### LD body



# AdrDT_Of_Offs32

Returns the DT address of the input or output with 32-bit offset

From the value at input/output **Offs**, the function generates a 16-bit address in the DT area at input/output **AdrDT**. This input has to be directly attached to a 32-bit input/output of a basic function. The value at input/output **Offs** represents the address offset within the DT data area. Using **GetPointer32** it can be transferred along with a value for the memory area to a user function or function block.




## Parameters

Input

**AdrDT** (DINT)

Yields 16-bit DT offset address

This pin must be connected to the input/output of a basic function for which the data type INT, WORD is allowed

Output

**Offs32** (WORD, INT, UINT)

32-bit offset for address DT0

## Remarks

- Remember, the execution of the basic function that uses the address created in the DT area is determined by assigning the function **Is_AreaDT** with the value for the memory area of the variable at EN-input.
- Only for LD and FBD Editors: Use “Input instruction” or “Output instruction” from the “Instructions” pane to insert the instruction required into the programming window.

## Example

### Using the user-defined function ReadFromIcCard in a program

In this example the user-defined function **ReadFromIcCard** is created that uses the functions **AdrDT_Of_Offs32_O**, **AdrFL_Of_Offs_O**, **Is_AreaDT** and **Is_AreaFL** in ladder diagram (LD) or the function block diagram (FBD).

## DUT

In the DUT Pool a structured data type is assigned in which the structure's various, non-boolean variables are declared.

DUT_NonBoolean [DUT]

| | Identifier | Type | Initial |
|---|---|---|---|
| 0 | Int1 | INT | 0 |
| 1 | ArrayOfInt1 | ARRAY [0..2] OF INT | [3(0)] |
| 2 | Word1 | WORD | 0 |
| 3 | ArrayOfDint1 | ARRAY [0..2] OF DINT | [3(0)] |
| 4 | Dword1 | DWORD | 0 |
| 5 | Dint1 | DINT | 0 |

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

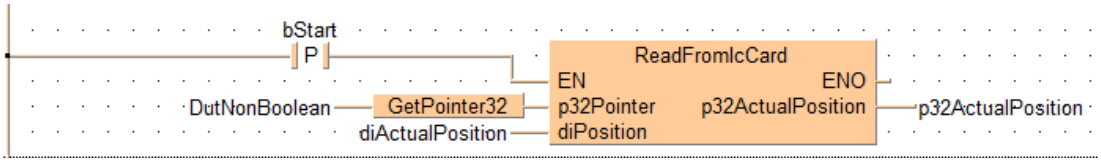
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | Activation of the function |
| 1 | VAR | DutNonBoolean | DUT_NonBoolean | | structured data type |
| 2 | VAR | diActualPosition | DINT | 0 | Beginning position from which data should... |
| 3 | VAR | p32ActualPosition | POINTER32 | | |

Here the variable **DutNonBoolean** of the data type assigned in the above DUT is declared. Assigning elements of the variable **DutNonBoolean** with values was not done in the POU header or body, since the values of the variable **DutNonBoolean** are overwritten after the function **ReadFromIcCard** is executed.

## POU body

When the variable **bStart** changes from FALSE to TRUE, the function **ReadFromIcCard** is carried out. The function reads values on the IC card beginning with address **diActualPosition** and writes the information to the variable **DutNonBoolean**. Do not forget that the IC card has to be appropriately formatted via the menu "Online" > "IC memory card manager" and that, if necessary, values beginning at address **diActualPosition** should be present on the IC card.

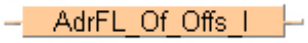
## LD body

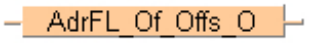


# AdrFL_Of_Offs

Returns the FL address of the input or output with offset

From the value at input/output **Offs**, the function generates a 16-bit address in the FL area at input/output **AdrFL**. This input/output has to be directly attached to a 16-bit input/output of a basic function. The value at input/output **Offs** represents the address offset within the FL data area. Using **GetPointer** it can be transferred along with a value for the memory area to a user function or function block.




## Parameters

Input

**AdrFL** (INT)

Yields 16-bit FL offset address

This pin must be connected to the input/output of a basic function for which the data type INT, WORD is allowed

Output

**Offs** (WORD, INT, UINT)

Offset for address FL0

## Remarks

- Remember, the execution of the basic function that uses the address created in the FL area is determined by assigning the function **Is_AreaFL** with the value for the memory area of the variable at EN-input.
- This function internally accesses one of the index registers that is also used for array calculations. If the FL area is not available for a given controller, the address DT0 is generated automatically instead of the FL address.
- The function **Is_AreaFL** always returns FALSE and thus hinders executing a command with a meaningless address. Hence, the user-defined functions and function blocks written with this function can be run on controllers that do not support the FL area.
- Only for LD and FBD Editors: Use “Input instruction” or “Output instruction” from the “Instructions” pane to insert the instruction required into the programming window.

## Example

### DUT

DUT_NonBoolean [DUT]

| | Identifier | Type | Initial |
|---|---|---|---|
| 0 | Int1 | INT | 0 |
| 1 | ArrayOfInt1 | ARRAY [0..2] OF INT | [3(0)] |
| 2 | Word1 | WORD | 0 |
| 3 | ArrayOfDint1 | ARRAY [0..2] OF DINT | [3(0)] |
| 4 | Dword1 | DWORD | 0 |
| 5 | Dint1 | DINT | 0 |

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

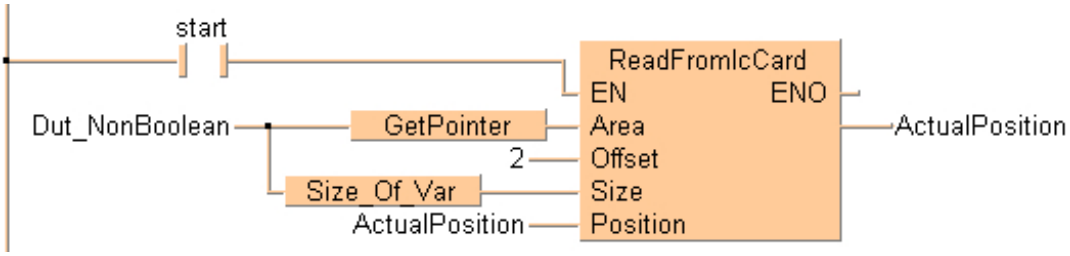
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activation of the function |
| 1 | VAR | DutNonBoolean | DUT_NonBoolean | | structured data type |
| 2 | VAR | Actual Position | DINT | 0 | Beginning position from which the data should be read to the IC card |

Here the variable **DutNonBoolean** of the data type assigned in the above DUT is declared. Assigning elements of the variable **DutNonBoolean** with values was not done in the POU header or body, since the values of the variable **DutNonBoolean** are overwritten after the function **ReadFromIcCard** is executed.

### POU body

When the variable **start** changes from FALSE to TRUE, the function **ReadFromIcCard** is carried out. The function reads values on the IC card beginning with address **ActualPosition** and writes the information to the variable **DutNonBoolean**. Do not forget that the IC card has to be appropriately formatted via the menu “Online” > “IC memory card manager...” and that, if necessary, values beginning at address **ActualPosition** should be present on the IC card.

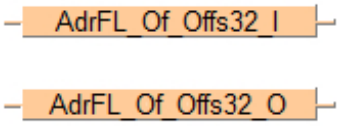
### LD body



# AdrFL_Of_Offs32

Returns the FL address of the input or output with 32-bit offset

From the value at input/output **Offs**, the function generates a 32-bit address in the FL area at input/output **AdrFL**. This input/output has to be directly attached to a 16-bit input/output of a basic function. The value at input/output **Offs** represents the address offset within the FL data area. Using **GetPointer32** it can be transferred along with a value for the memory area to a user function or function block.




## Parameters

Input

**AdrFL** (DINT)

Yields 16-bit FL offset address

This pin must be connected to the input/output of a basic function for which the data type INT, WORD is allowed

Output

**Offs32** (WORD, INT, UINT)

32-bit offset for address FL0

## Remarks

- Remember, the execution of the basic function that uses the address created in the FL area is determined by assigning the function **Is_AreaFL** with the value for the memory area of the variable at EN-input.
- This function internally accesses one of the index registers that is also used for array calculations. If the FL area is not available for a given controller, the address DT0 is generated automatically instead of the FL address.
- The function **Is_AreaFL** always returns FALSE and thus hinders executing a command with a meaningless address. Hence, the user-defined functions and function blocks written with this function can be run on controllers that do not support the FL area.
- Only for LD and FBD Editors: Use “Input instruction” or “Output instruction” from the “Instructions” pane to insert the instruction required into the programming window.

## Example

### DUT

DUT_NonBoolean [DUT]

| | Identifier | Type | Initial |
|---|---|---|---|
| 0 | Int1 | INT | 0 |
| 1 | ArrayOfInt1 | ARRAY [0..2] OF INT | [3(0)] |
| 2 | Word1 | WORD | 0 |
| 3 | ArrayOfDint1 | ARRAY [0..2] OF DINT | [3(0)] |
| 4 | Dword1 | DWORD | 0 |
| 5 | Dint1 | DINT | 0 |

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

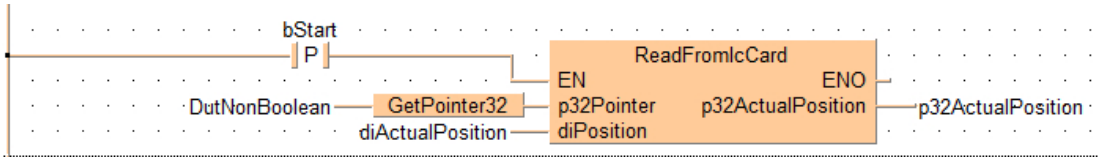
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | Activation of the function |
| 1 | VAR | DutNonBoolean | DUT_NonBoolean | | structured data type |
| 2 | VAR | diActualPosition | DINT | 0 | Beginning position from which data should... |
| 3 | VAR | p32ActualPosition | POINTER32 | | |

Here the variable **DutNonBoolean** of the data type assigned in the above DUT is declared. Assigning elements of the variable **DutNonBoolean** with values was not done in the POU header or body, since the values of the variable **DutNonBoolean** are overwritten after the function **ReadFromIcCard** is executed.

### POU body

When the variable **bStart** changes from FALSE to TRUE, the function **ReadFromIcCard** is carried out. The function reads values on the IC card beginning with address **diActualPosition** and writes the information to the variable **DutNonBoolean**. Do not forget that the IC card has to be appropriately formatted via the menu “Online” > “IC memory card manager” and that, if necessary, values beginning at address **diActualPosition** should be present on the IC card.

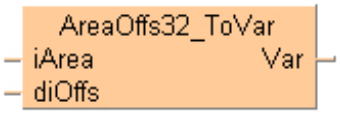
### LD body



# AreaOffs32_ToVar

Copies the content of an address to a variable with 32-bit offset

This function copies number of words defined by the size of the variable **Var** to the variable **Var** from the address determined by the memory area **iArea** and the address offset **diOffs**.




## Parameters

Input

**iArea** (INT)

Value for the memory area[1)]
If **iArea** is a variable, then it must be located in the memory area DT or FL. This should be checked using a function that incorporates the FP Tool Library's functions **Is_AreaDT** or **Is_AreaFL**.

**diOffs** (DINT)

Offset for the starting 32-bit address of the memory area

Output

**Var** (ANY_SIMPLE_NOT_BOOL)

Variable to which the address's content is copied

## Remarks

The values for **iArea** and **diOffs** can be supplied via the function **GetPointer32**. The value for **iArea** has to lie in the DT orFL area.

## Value for the memory area[1)]

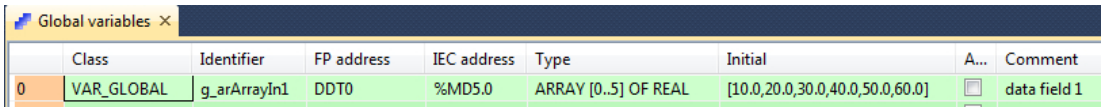
| | |
|---|---|
| Flag memory areas | **SYS_MEMORY_AREA_R** |
| | **SYS_MEMORY_AREA_L** |
| | **SYS_MEMORY_AREA_X** |
| | **SYS_MEMORY_AREA_Y** |
| Flags | **SYS_MEMORY_AREA_WR** |
| Set value timer/counter | **SYS_MEMORY_AREA_SV** |
| Elapsed value timer/counter | **SYS_MEMORY_AREA_EV** |

| | |
|---|---|
| Data registers | **SYS_MEMORY_AREA_DT** |
| Link flags | **SYS_MEMORY_AREA_WL** |
| Link registers | **SYS_MEMORY_AREA_LD** |
| File registers | **SYS_MEMORY_AREA_FL** |
| Input registers | **SYS_MEMORY_AREA_WX** |
| Output registers | **SYS_MEMORY_AREA_WY** |

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.




Here the variables **RealArrayIn1** and **RealArrayIn2** are assigned. They were declared in different memory areas on purpose to illustrate the efficacy of the function block **CalcSum_REAL**. The variables could just as well be declared In the POU header of the program **Program_CalcSums**.

### POU header

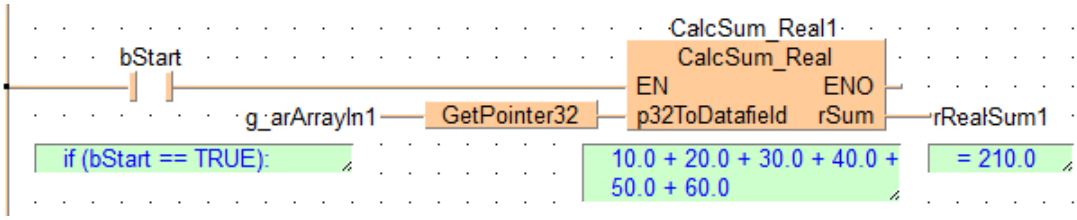
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_arArrayIn1 | ARRAY [0..5] OF RE... | [10.0,20.0,... | data field 1 |
| 1 | VAR | bStart | BOOL | FALSE | activation |
| 2 | VAR | rRealSum1 | REAL | 0.0 | sum of the elements of data field 1 |
| 3 | VAR | CalcSum_Real1 | CalcSum_Real | | instance of the user function block |

### POU body

When the variable **bSart** is set to TRUE, the function block **CalcSum_REAL** is carried out. It calculates the sum of all elements of the data field **RealArrayIn1** and writes the result to the variable **RealSum1**.

### LD body




## Example programming the user-defined function block CalcSum_REAL

### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_INPUT | p32ToDatafield | POINTER32 | | pointer on the data field for calculation |
| 1 | VAR_OUTPUT | rSum | REAL | 0.0 | calculate sum |
| 2 | VAR | Sum_Operand | REAL | 0.0 | inernal calculation of the sum |
| 3 | VAR | In_Offs_Intern | DINT | 0 | internal offset |
| 4 | VAR | Idx | INT | 0 | index |

### LD body

- Network 1:

  If the data field delivered does not lie in the DT or FL area, the ENO of the function block is set to FALSE and the processing ended.

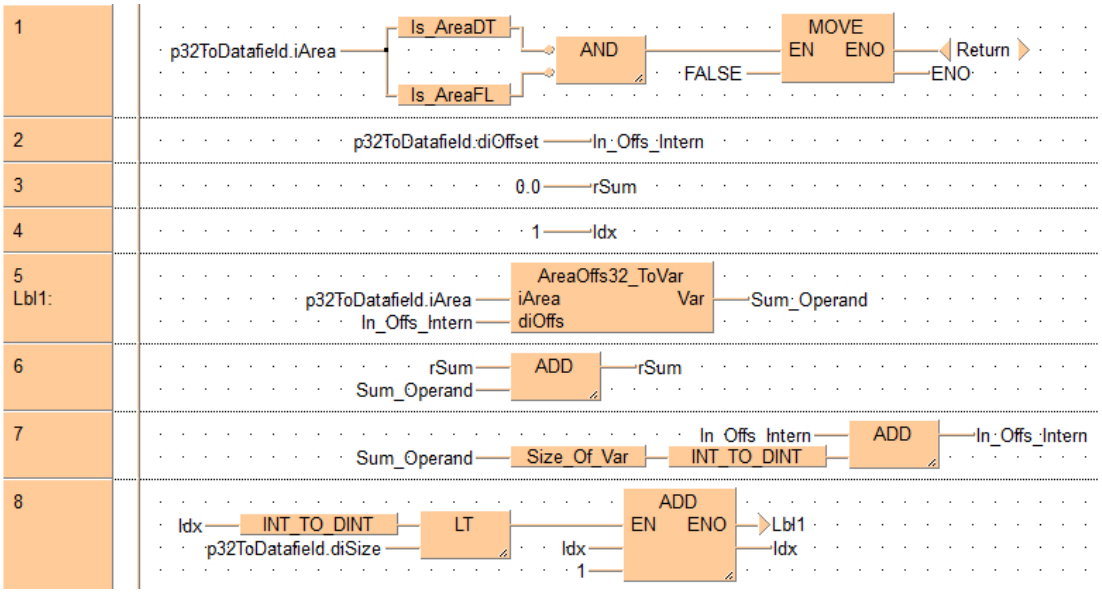
- Network 2: 4:

  Here the initializations for calculating the sum that follows is dealt with. Since variables of the class VAR_INPUT cannot be changed directly, the input variable **In_Offset** has to be copied to the internal variable **In_Offs_intern**.

  see also: Var_ToAreaOffs32

- Network 5: 8:

  Loop for the sum calculation.




## Related topics

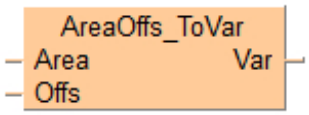
Further example projects (directory "Samples" of FPWIN Pro installation):

**Var_ToAreaOffs32**

# AreaOffs_ToVar

Copies the content of an address to a variable with 16-bit offset

This function copies number of words defined by the size of the variable **Var** to the variable **Var** from the address determined by the memory area **Area** and the address offset **Offs**.




## Parameters

Input

**iArea** (INT)

Value for the memory area[1)]
If **Area** is a variable, then it must be located in the memory area DT or FL. This should be checked using a function that incorporates the FP Tool Library's functions **Is_AreaDT** or **Is_AreaFL**.

**Offs** (INT)

Offset for the starting 16-bit address of the memory area

Output

**Var** (ANY_SIMPLE_NOT_BOOL)

Variable to which the address's content is copied

## Remarks

The values for **Area** and **Offs** can be supplied via the function **GetPointer**. The value for **Area** has to lie in the DT orFL area.

## Value for the memory area[1)]

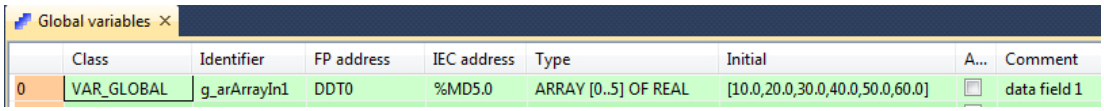
| | |
|---|---|
| Flag memory areas | **SYS_MEMORY_AREA_R** |
| | **SYS_MEMORY_AREA_L** |
| | **SYS_MEMORY_AREA_X** |
| | **SYS_MEMORY_AREA_Y** |
| Flags | **SYS_MEMORY_AREA_WR** |
| Set value timer/counter | **SYS_MEMORY_AREA_SV** |
| Elapsed value timer/counter | **SYS_MEMORY_AREA_EV** |

| | |
|---|---|
| Data registers | **SYS_MEMORY_AREA_DT** |
| Link flags | **SYS_MEMORY_AREA_WL** |
| Link registers | **SYS_MEMORY_AREA_LD** |
| File registers | **SYS_MEMORY_AREA_FL** |
| Input registers | **SYS_MEMORY_AREA_WX** |
| Output registers | **SYS_MEMORY_AREA_WY** |

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.




Here the variable **g_arArrayIn1** is assigned. The variable could just as well be declared In the POU header of the program **Program_CalcSums**.

### POU header

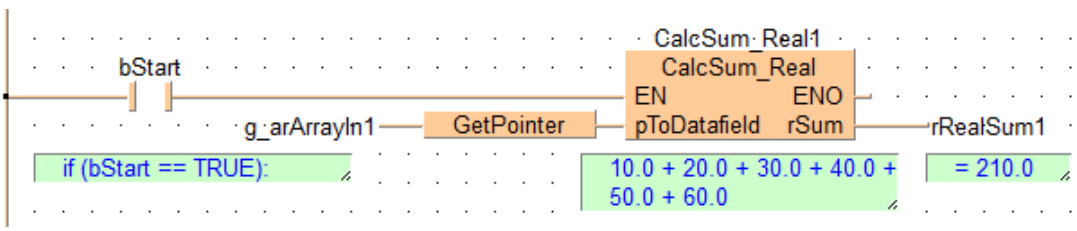
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_arArrayIn1 | ARRAY [0..5] OF RE... | [10.0,20.0,... | data field 1 |
| 1 | VAR | bStart | BOOL | FALSE | activation |
| 2 | VAR | rRealSum1 | REAL | 0.0 | sum of the elements of data field 1 |
| 3 | VAR | CalcSum_Real1 | CalcSum_Real | | instance of the user function block |

### POU body

When the variable **bStart** is set to TRUE, the function block **CalcSum_REAL** is carried out. It calculates the sum of all elements of the data field **g_arArrayIn1** and writes the result to the variable **rRealSum1**.

### LD body



# Example programming the user-defined function block CalcSum_REAL

## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_INPUT | pToDatafield | POINTER | | pointer on the data field for calculation |
| 1 | VAR_OUTPUT | rSum | REAL | 0.0 | calculate sum |
| 2 | VAR | Sum_Operand | REAL | 0.0 | inernal calculation of the sum |
| 3 | VAR | In_Offs_Intern | INT | 0 | internal offset |
| 4 | VAR | Idx | INT | 0 | index |

## LD body

- Network 1:

  If the data field delivered does not lie in the DT or FL area, the ENO of the function block is set to FALSE and the processing ended.

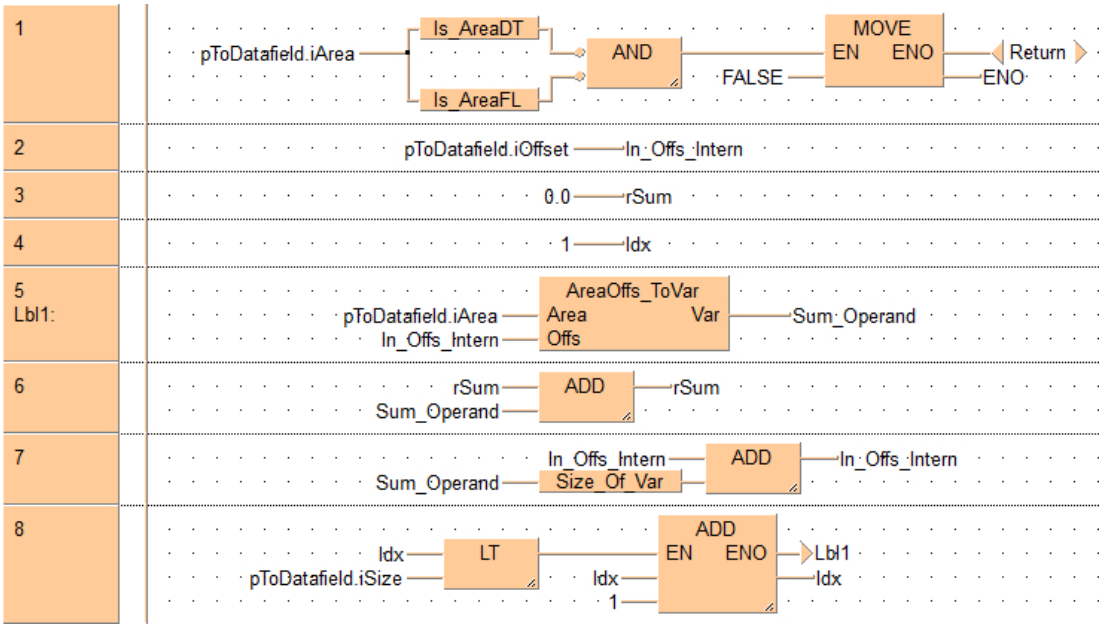
- Network 2: 4:

  Here the initializations for calculating the sum that follows is dealt with. Since variables of the class VAR_INPUT cannot be changed directly, the input variable **In_Offset** has to be copied to the internal variable **In_Offs_intern**.

  see also: Var_ToAreaOffs

- Network 5: 8:

  Loop for the sum calculation.



## Related topics

Further example projects (directory "Samples" of FPWIN Pro installation):

**Var_ToAreaOffs**

Calculates the product of two-dimensional arrays

Calculate the sum of all array elements

Calculates the trace of a three-dimensional array

# GetPointer

provides pointer information

The **GetPointer** function provides variables' pointer information in a predefined data unit type (DUT) of the data type POINTER which can be used by functions or function blocks. With such pointer information and the corresponding pointer functions, functions or function blocks can directly read from or write to the external data's address area.


## Parameters

Input

**Unnamed input** (ANY)

Input variable of any data type that must be located in the DT, FL or Y memory area.

Output

**Unnamed output** (POINTER)

The pointer information of the input variable.

## Remarks

User-defined functions or user-defined function blocks can use the pointer information as follows:

- The pointer element **iSize** can specify the number of values, i.e. (number of elements = Pointer.iSize / Size_Of_Var (DataElement))
- With corresponding pointer functions **AreaOffs_ToVar** and**Var_ToAreaOffs** and with the pointer elements **iArea** and **iOffset**, you can directly read from or write to the external data's address area.
- With the memory area instructions **Is_AreaDT** or **Is_AreaFL** and with the address instructions **AdrDT_Of_Offs** or **AdrFL_Of_Offs**, you can also use FP instructions with the external data's address areas.

The input variable must be located in the memory area DT or FL for the following reasons:

- because the memory area instructions **Is_AreaDT** or **Is_AreaFL** and the address instructions **AdrDT_Of_Offs** or **AdrFL_Of_Offs** work only with these memory areas.
- because the pointer instructions **AreaOffs_ToVar** and**Var_ToAreaOffs** work correct only within these memory areas.

This instruction replaces the instruction **AreaOffs_OfVar**.

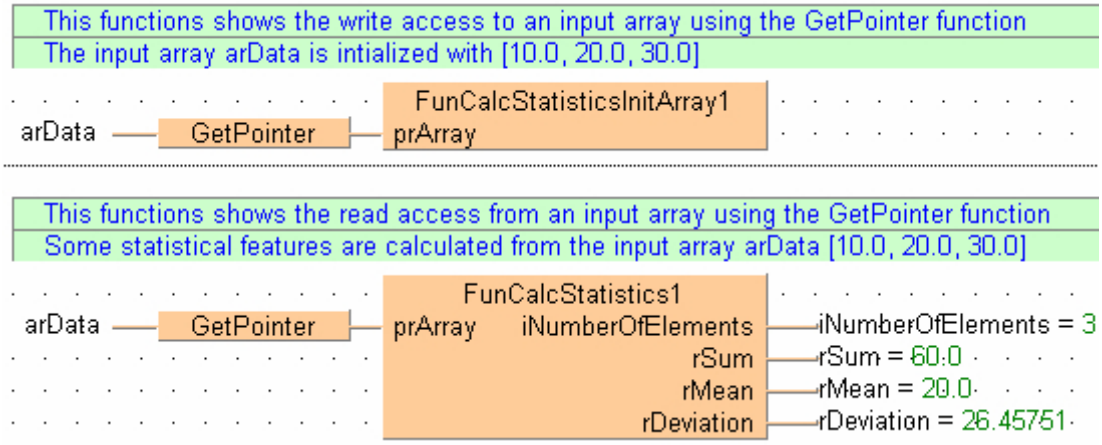
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | arData | ARRAY [0..2] OF REAL | [3(0.0)] |
| 1 | VAR | iNumberOfElements | INT | 0 |
| 2 | VAR | rSum | REAL | 0.0 |
| 3 | VAR | rMean | REAL | 0.0 |
| 4 | VAR | rDeviation | REAL | 0.0 |

### LD body




## FunCalcStatisticsInitArray1Example

### POU header

This function initializes the data being pointed at by **prArray** with increasing values of 10, 20, 30...

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_INPUT | prArray | POINTER32 | |
| 1 | VAR | di | DINT | 0 |
| 2 | VAR | diNumberOfElements | DINT | 0 |
| 3 | VAR | diOffset | DINT | 0 |
| 4 | VAR | rCurrentElement | REAL | 0.0 |

## POU header

The user-defined function **FunCalcStatistics1** calculates statistics from the values being pointed at by **prArray**. Internally the instruction AreaOffs_ToVar is used to read from the external data's address areas.

All input and output variables used for programming this function have been declared in the POU header.

The same POU header is used for all programming languages.
**FunCalcStatistics1**

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_INPUT | prArray | POINTER32 | |
| 1 | VAR_OUTPUT | diNumberOfElements | DINT | 0 |
| 2 | VAR_OUTPUT | rSum | REAL | 0.0 |
| 3 | VAR_OUTPUT | rMean | REAL | 0.0 |
| 4 | VAR_OUTPUT | rDeviation | REAL | 0.0 |
| 5 | VAR | di | DINT | 0 |
| 6 | VAR | diOffset | DINT | 0 |
| 7 | VAR | rCurrentElement | REAL | 0.0 |
| 8 | VAR | rNumberOfElements | REAL | 0.0 |
| 9 | VAR | rSquareSum | REAL | 0.0 |

# GetPointer32

Provides pointer information

The **GetPointer32** function provides variables' 32-bit pointer information in a predefined data unit type (DUT) of the data type POINTER32 which can be used by functions or function blocks. With such pointer information and the corresponding pointer functions, functions or function blocks can directly read from or write to the external data's address area.


## Parameters

Input

**Unnamed input** (ANY)

Input variable of any data type that must be located in the DT, FL or Y memory area.

Output

**Unnamed output** (POINTER32)

The pointer information of the input variable.

## Remarks

User-defined functions or user-defined function blocks can use the pointer information as follows:

- The pointer element **iSize** can specify the number of values, i.e. (number of elements = Pointer.iSize / **Size_Of_Var** (DataElement))
- With corresponding pointer functions **AreaOffs32_ToVar** and **Var_ToAreaOffs32** and with the pointer elements **iArea** and **diOffset**, you can directly read from or write to the external data's address area.
- With the memory area instructions **Is_AreaDT** or **Is_AreaFL** and with the address instructions **AdrDT_Of_Offs32** or **AdrFL_Of_Offs32**, you can also use FP instructions with the external data's address areas.

The input variable must be located in the memory area DT or FL for the following reasons:

- because the memory area instructions **Is_AreaDT** or **Is_AreaFL** and the address instructions **AdrDT_Of_Offs32** or **AdrFL_Of_Offs32** work only with these memory areas.
- because the pointer instructions **AreaOffs32_ToVar** and **Var_ToAreaOffs32** work correct only within these memory areas.

This instruction replaces the instruction **AreaOffs_OfVar**.

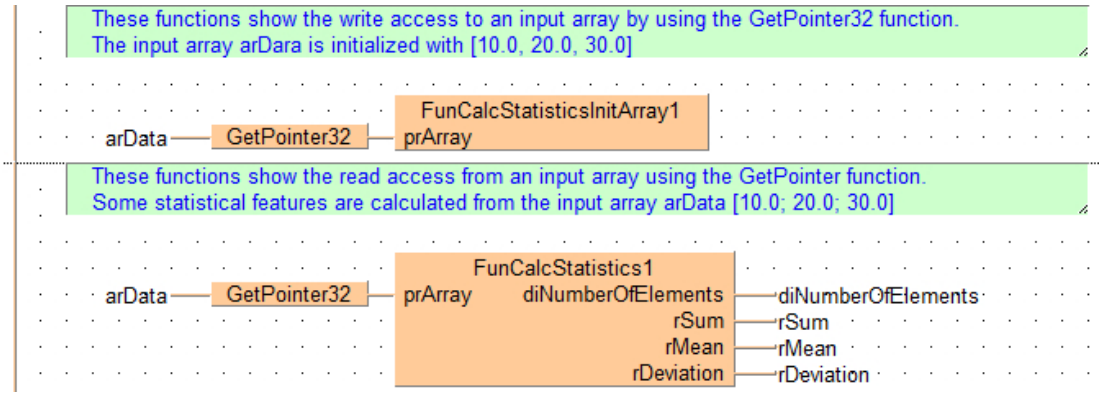
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | arData | ARRAY [0..2] OF REAL | [10.0,20.0,... |
| 1 | VAR | diNumberOfElements | DINT | 0 |
| 2 | VAR | rSum | REAL | 0.0 |
| 3 | VAR | rMean | REAL | 0.0 |
| 4 | VAR | rDeviation | REAL | 0.0 |

### LD body




## FunCalcStatisticsInitArray1Example:

### POU header

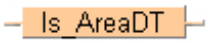
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_INPUT | prArray | POINTER32 | |
| 1 | VAR | di | DINT | 0 |
| 2 | VAR | diNumberOfElements | DINT | 0 |
| 3 | VAR | diOffset | DINT | 0 |
| 4 | VAR | rCurrentElement | REAL | 0.0 |

# Is_AreaDT

Yields TRUE if the memory area of a variable is a DT area

From the input value, this function generates TRUE or FALSE at the output, depending on whether the value corresponds to the DT area or not. The input value can be transferred along with the address offset of a variable to a user function or function block via the function **GetPointer** (page 1432).




## Parameters

Input

**Unnamed input** (INT)

Input variable

Output

**Unnamed output** (BOOL)

Yields TRUE if the value of the input variable is 5 otherwise FALSE

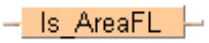
## Remarks

By assigning **Is_AreaDT** to an **EN** input of a basic function with an address in the **AdrDT_Of_Offs** area created by **AdrDT_Of_Offs** (page 1415), its execution can be adjusted accordingly.

# Is_AreaFL

Yields TRUE if the memory area of a variable is a FL area

From the input value, this function generates TRUE or FALSE at the output, depending on whether the value corresponds to the FL area or not. The input value can be transferred along with the address offset of a variable to a user function or function block via the function **GetPointer**.




## Parameters

Input

**Unnamed input** (INT)

Input variable

Output

**Unnamed output** (BOOL)

Yields TRUE if the value of the input variable is 9, otherwise FALSE

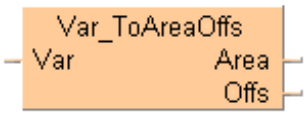
## Remarks

By assigning **Is_AreaFL** to an **EN** input of a basic function with an address in the **AdrFL_Of_Offs** area created by **AdrFL_Of_Offs**, its execution can be adjusted accordingly.

# Var_ToAreaOffs

Copies a variable value to an address area

This function copies the value of the variable at input **Var** to the address area defined by the memory area **Area** and the address offset **Offs**. The size of the variable at input **Var** thereby determines the number of data copied.




## Parameters

Input

**Var** (ANY_SIMPLE_NOT_BOOL)

Variable whose value is copied to an address

Output

**iArea** (INT)

Value for the memory area[1)]
If **iArea** is a variable, then it must be located in the memory area DT or FL. This should be checked using a function that incorporates the FP Tool Library's functions **Is_AreaDT** or **Is_AreaFL**.

**Offs** (INT)

Offset for the starting 16-bit address of the memory area

## Remarks

The values for **iArea** and **diOffs** can be supplied via the function **GetPointer32**. The value for **iArea** has to lie in the DT orFL area.

## Value for the memory area[1)]

| | |
|---|---|
| Flag memory areas | **SYS_MEMORY_AREA_R** |
| | **SYS_MEMORY_AREA_L** |
| | **SYS_MEMORY_AREA_X** |
| | **SYS_MEMORY_AREA_Y** |
| Flags | **SYS_MEMORY_AREA_WR** |
| Set value timer/counter | **SYS_MEMORY_AREA_SV** |

| | |
|---|---|
| Elapsed value timer/counter | **SYS_MEMORY_AREA_EV** |
| Data registers | **SYS_MEMORY_AREA_DT** |
| Link flags | **SYS_MEMORY_AREA_WL** |
| Link registers | **SYS_MEMORY_AREA_LD** |
| File registers | **SYS_MEMORY_AREA_FL** |
| Input registers | **SYS_MEMORY_AREA_WX** |
| Output registers | **SYS_MEMORY_AREA_WY** |

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

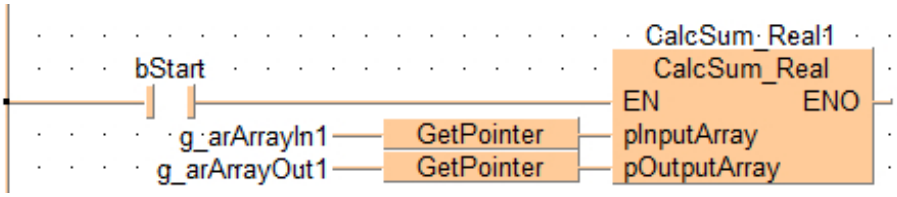
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activation |
| 1 | VAR | g_arArrayOut1 | ARRAY [0..5] OF REAL | [6(0.0)] | data field with cumulative sum |
| 2 | VAR | g_arArrayIn1 | ARRAY [0..5] OF REAL | [10.0,20.0,... | data field 1 |
| 3 | VAR | CalcSum_Real1 | CalcSum_Real | | instance of user function block |

### POU body

When the variable **bStart** is set to TRUE, the function block **CalcSum_REAL** is carried out. It calculates the sum of all elements of the data field **g_arArrayIn1** and writes the result to the data field **g_arArrayOut1**. In this way the first element of **g_arArrayOut1** receives the first element of **g_arArrayIn1**. The second element of **g_arArrayOut1** contains the sums of the first and second element of **g_arArrayIn1**.The third element of **g_arArrayOut1** contains the sums of the first, second and third element of **g_arArrayIn1**.....

Remember that the function block **CalcSum_REAL** can calculate its function independently of the memory areas in which both data fields can be stored and independently of the number of elements in the data fields. If you wish to assign an address for the data field, you have to declare the data field in the global variable list.Both data fields must possess the same number of elements.

### LD body



# Example programming the user-defined function block CalcSum_REAL

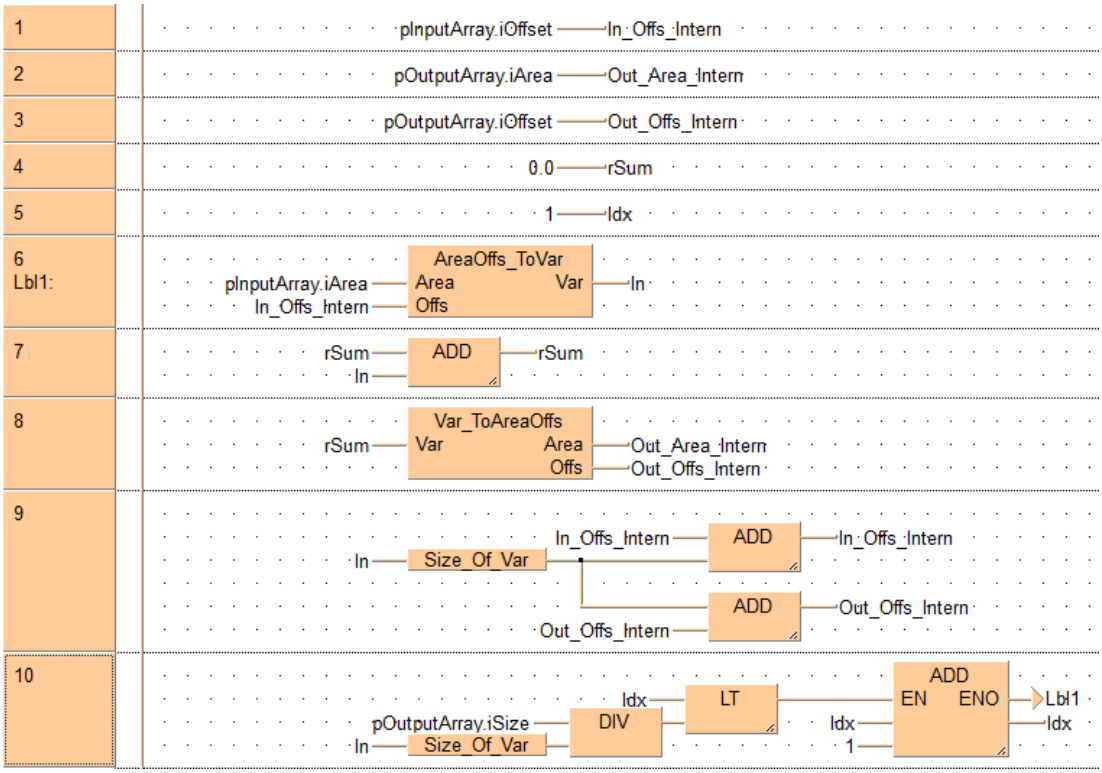
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_INPUT | pInputArray | POINTER | | pointer on the data field input |
| 1 | VAR_INPUT | pOutputArray | POINTER | | pointer on the data field output |
| 2 | VAR | rSum | REAL | 0.0 | calculate sum |
| 3 | VAR | In | REAL | 0.0 | |
| 4 | VAR | In_Offs_Intern | INT | 0 | internal offset |
| 5 | VAR | Out_Area_Intern | INT | 0 | memory area intern data field output |
| 6 | VAR | Out_Offs_Intern | INT | 0 | offset intern data field output |
| 7 | VAR | Idx | INT | 0 | index |

## LD body

- Network 1–5:
  here the initializations for calculating the sum that follows is dealt with.
- Network 6:
  copies value of an element from the input data field.
- Network 7:
  adds this value to the current sum.
- Network 8:
  copies current sum to an element of the output data field.
- Network 9:
  calculation of the offset for the next data field element of the input data field and output data field.
- Network 10: jump to the label (network 6) until all elements of the data field have been processed.




## Related topics

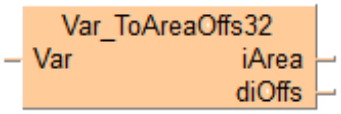
Further example projects (directory "Samples" of FPWIN Pro installation):

- Calculates the product of two-dimensional arrays

# Var_ToAreaOffs32

Copies a variable value to an address area

This function copies the value of the variable at input **Var** to the address area defined by the memory area **iArea** and the 32-bit address offset **diOffs**. The size of the variable at input **Var** thereby determines the number of data copied.




## Parameters

Input

**Var** (ANY_SIMPLE_NOT_BOOL)

Variable whose value is copied to an address

Output

**iArea** (INT)

Value for the memory area[1)]
If **iArea** is a variable, then it must be located in the memory area DT or FL. This should be checked using a function that incorporates the FP Tool Library's functions **Is_AreaDT** or **Is_AreaFL**.

**diOffs** (DINT)

Offset for the starting 32-bit address of the memory area

## Remarks

The values for **iArea** and **diOffs** can be supplied via the function **GetPointer32**. The value for **iArea** has to lie in the DT orFL area.

## Value for the memory area[1)]

| | |
|---|---|
| Flag memory areas | **SYS_MEMORY_AREA_R** |
| | **SYS_MEMORY_AREA_L** |
| | **SYS_MEMORY_AREA_X** |
| | **SYS_MEMORY_AREA_Y** |
| Flags | **SYS_MEMORY_AREA_WR** |
| Set value timer/counter | **SYS_MEMORY_AREA_SV** |

| | |
|---|---|
| Elapsed value timer/counter | **SYS_MEMORY_AREA_EV** |
| Data registers | **SYS_MEMORY_AREA_DT** |
| Link flags | **SYS_MEMORY_AREA_WL** |
| Link registers | **SYS_MEMORY_AREA_LD** |
| File registers | **SYS_MEMORY_AREA_FL** |
| Input registers | **SYS_MEMORY_AREA_WX** |
| Output registers | **SYS_MEMORY_AREA_WY** |

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

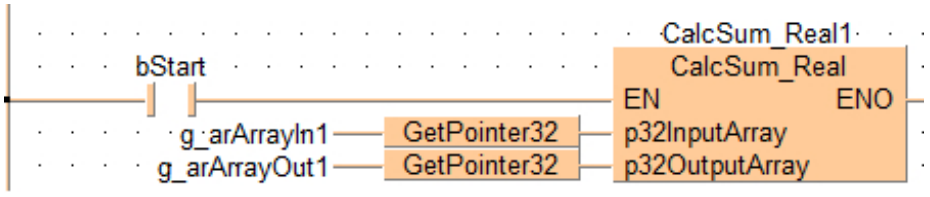
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activation |
| 1 | VAR | g_arArrayOut1 | ARRAY [0..5] OF REAL | [6(0.0)] | data field with cumulative sum |
| 2 | VAR | g_arArrayIn1 | ARRAY [0..5] OF REAL | [10.0,20.0,... | data field 1 |
| 3 | VAR | CalcSum_Real1 | CalcSum_Real | | instance of user function block |

### POU body

When the variable **bStart** is set to TRUE, the function block **CalcSum_REAL** is carried out. It calculates the sum of all elements of the data field **g_arArrayIn1** and writes the result to the data field **g_arArrayOut1**. In this way the first element of **g_arArrayOut1** receives the first element of **g_arArrayIn1**. The second element of **g_arArrayOut1** contains the sums of the first and second element of **g_arArrayIn1**.The third element of **g_arArrayOut1** contains the sums of the first, second and third element of **g_arArrayIn1**.....

Remember that the function block **CalcSum_REAL** can calculate its function independently of the memory areas in which both data fields can be stored and independently of the number of elements in the data fields. If you wish to assign an address for the data field, you have to declare the data field in the global variable list.Both data fields must possess the same number of elements.

### LD body



# Example: programming the user-defined function block CalcSum_REAL

## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_INPUT | p32InputArray | POINTER32 | | pointer on the data field input |
| 1 | VAR_INPUT | p32OutputArray | POINTER32 | | pointer on the data field output |
| 2 | VAR | rSum | REAL | 0.0 | calculate sum |
| 3 | VAR | In | REAL | 0.0 | |
| 4 | VAR | In_Offs_Intern | DINT | 0 | internal offset |
| 5 | VAR | Out_Area_Intern | INT | 0 | memory area intern data field output |
| 6 | VAR | Out_Offs_Intern | DINT | 0 | offset intern data field output |
| 7 | VAR | Idx | INT | 0 | index |

## LD body

- Network 1–5:

  here the initializations for calculating the sum that follows is dealt with.
- Network 6:

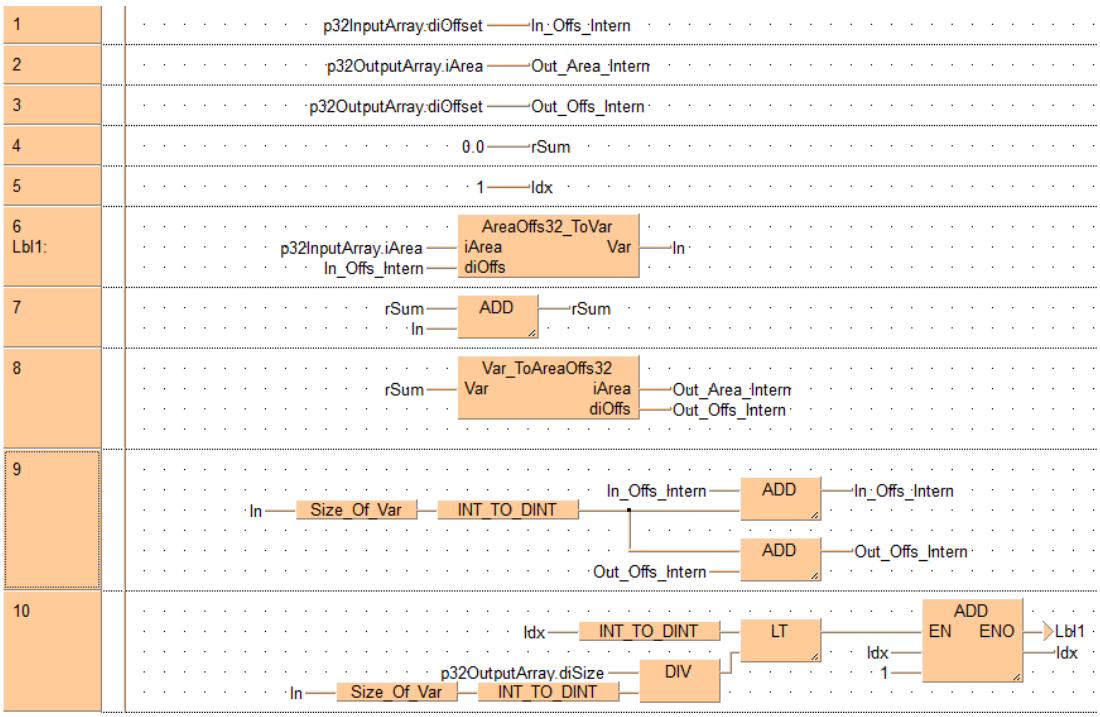
  copies value of an element from the input data field.
- Network 7:

  adds this value to the current sum.
- Network 8:

  copies current sum to an element of the output data field.
- Network 9:

  calculation of the offset for the next data field element of the input data field and output data field.
- Network 10:

  jump to the label (network 6) until all elements of the data field have been processed.




## Related topics

Further example projects (directory "Samples" of FPWIN Pro installation):

- Calculates the product of two-dimensional arrays

# 24 Process control instructions

# CAL

Unconditional Call

After the execution of the **CAL** command, the function block defined in the operand field is called. After finishing processing the function block, the main program will be executed with the command following the **CAL** command.

## Remarks

Valid operands for this operator must be of one of the following data types: Any valid function block identifier.

## Example

### POU header

The same POU header is used for all programming languages. All input and output variables used for programming this function have been declared in the POU header.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | Counter | CTU | | Counter instance |
| 1 | VAR | clock | BOOL | FALSE | upward counter input |
| 2 | VAR | reset | BOOL | FALSE | Resets the counter |
| 3 | VAR | signal_output | BOOL | FALSE | Counter output, TRUE if counter reaches preset value |

### IL body

```
1   (*counting always enabled*)
    CAL     Counter(CU:= clock ,
            RESET:= reset ,
            PV:= 10 ,
            Q:= signal_output)
```

# CALC

Conditional Call

If the content of the accumulator is TRUE, the function block defined in the operand field is called. After finishing processing the function block, the main program will be executed with the command following the **CALC** command.

## Remarks

- Valid operands for this operator must be of one of the following data types: Any valid function block identifier.
- Operator only available in IL programming language.

## Example

In this example the standard operator CALC is programmed in instruction list (IL).

### POU header

In the POU header, all input and output variables are declared that are used for programming this function.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | Counter | CTU | | Counter instance |
| 1 | VAR | Enable_Counting | BOOL | FALSE | enables counting |
| 2 | VAR | clock | BOOL | FALSE | upward counter input |
| 3 | VAR | reset | BOOL | FALSE | resets the counter |
| 4 | VAR | signal_output | BOOL | FALSE | Counter output, TRUE if counter reaches |

### IL body

```
1
        LD      Disable_counting
        CALC    Counter(CU:= clock,
                        RESET:= reset,
                        PV:= 10,
                        Q:= signal_output,
                        CV:= icurrentvalue)
```

# CALCN

Conditional Call NOT

If the content of the accumulator is FALSE, the function block defined in the operand field is called. After finishing processing the function block, the main program will be executed with the command following the **CALCN** command.

## Remarks

- Valid operands for this operator must be of one of the following data types: Any valid function block identifier.
- Operator only available in IL programming language.

## Example

The same POU header is used for all programming languages. POU header
In the POU header, all input and output variables are declared that are used for programming this function.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | Counter | CTU | | Counter instance |
| 1 | VAR | Disable_counting | BOOL | FALSE | Disables counting |
| 2 | VAR | clock | BOOL | FALSE | upward counter input |
| 3 | VAR | reset | BOOL | FALSE | Resets the counter |
| 4 | VAR | signal_output | BOOL | FALSE | Counter output, TRUE if counter reaches preset value |

IL body

```
1       LD      Disable_counting
        CALCN   Counter(CU:= clock,
                        RESET:= reset,
                        PV:= 10,
                        Q:= signal_output,
                        CV:= icurrentvalue)
```

# JMP

Unconditional Jump

The program execution is continued at the label defined in the operand field.

## Remarks

- Valid operands for this operator must be of one of the following data types: Any existing label in the program is valid.
- Operator only available in IL programming language.
- var_1, var_2, var_3, var_4 and var_5 must be of numeric data types in this example. LABEL1 and LABEL2 must be valid labels within the program. A colon must follow the label itself, in the **JMP** statement, however, the colon after the label name has to be omitted.
- Labels can only be inserted at the beginning of a network. A label can be inserted at the beginning of any network, even if it is not jumped to (e.g. LABEL0 in this code sample fragment).

## Example

POU header
In the POU header, all input and output variables are declared that are used for programming this function.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | var_1 | INT | 0 | Input_1 |
| 1 | VAR | var_2 | INT | 0 | Input_2 |
| 2 | VAR | var_3 | INT | 0 | Input_3 |
| 3 | VAR | var_4 | INT | 0 | Input_4 |
| 4 | VAR | var_5 | INT | 0 | Output |

IL body

| | | | |
|---|---|---|---|
| | | | (* Beginning of network 0 *) |
| LABEL0: | LD | **var_1** | (* Load **var_1** in accu *) |
| | ADD | **var_2** | (* Add **var_2** to accu; result is stored in accu *) |
| | JMP | LABEL1 | (* Continue program execution at position marked by LABEL1 *) |
| | | | |
| | | | (* Beginning of network 1 *) |
| LABEL1: | MUL | **var_3** | (* Multiply accu by **var_3** *) |
| | SUB | **var_4** | (* Subtract **var_4** from accu; result is store in accu *) |

| | | | |
|---|---|---|---|
| | ST | **var_5** | (* Store accu in **var_5** *) |
| | JMP | LABEL2 | (* Continue program execution at position marked by LABEL2 *) |
| LABEL2: | | | |

# JMPC

Conditional Jump

If the content of the accumulator is TRUE, the program execution is continued at the label defined in the operand field.

Valid operands for this operator must be of one of the following data types: Any existing label in the program is valid. The value in the accumulator must be of data type BOOL.

Note

- Operator only available in IL programming language.
- var_1 and var_2 can be of any data type since comparison is defined for any data type. var_3, var_4 and var_5 must be of numeric data types according to the operations performed with them. LABEL1 and LABEL2 must be valid labels within the program. The jump is only executed if the accu holds the value TRUE. Otherwise, program execution continues with the statement following the **JMPC** command.
- A colon must follow the label itself, in the **JMPC** statement, however, the colon after the label name has to be omitted.
- Labels can only be inserted at the beginning of a network. A label can be inserted at the beginning of any network, even if it is not jumped to. (e.g. LABEL0 in this code sample fragment).

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | var_1 | INT | 0 | Input_1 |
| 1 | VAR | var_2 | INT | 0 | Input_2 |
| 2 | VAR | var_3 | INT | 0 | Input_3 |
| 3 | VAR | var_4 | INT | 0 | Input_4 |
| 4 | VAR | var_5 | INT | 0 | Output |

### IL body

| | | | |
|---|---|---|---|
| | | | (* Beginning of network 0 *) |
| LABEL0: | LD | **var_1** | (* Load **var_1** in accu *) |

| | | | |
|---|---|---|---|
| | EQ | **var_2** | (* Compare accu with **var_2**; store result of comparison in accu; result is of type BOOL*) |
| | JMPC | LABEL1 | (* If accu contains TRUE continue program execution at position LABEL1 *) |
| | | | |
| | | | (* Beginning of network 1 *) |
| LABEL1: | MUL | **var_3** | (* Multiply accu by **var_3** *) |
| | SUB | **var_4** | (* Subtract **var_4** from accu; result is store in accu *) |
| | ST | **var_5** | (* Store accu in **var_5***) |

# JMPCN

Conditional Jump NOT

If the content of the accumulator is FALSE, the program execution is continued at the label defined in the operand field.

Any existing label in the program is valid. The value in the accumulator must be of data type BOOL.

Note

- Operator only available in IL programming language.
- var_1 and var_2 can be of any data type since comparison is defined for any data type. var_3, var_4 and var_5 must be of numeric data types according to the operations performed with them. LABEL1 and LABEL2 must be valid labels within the program. The jump is only executed if the accu holds the value FALSE. Otherwise, program execution continues with the statement following the JMPCN command.
- A colon must follow the label itself, in the JMPCN statement, however, the colon after the label name has to be omitted.
- Labels can only be inserted at the beginning of a network. A label can be inserted at the beginning of any network, even if it is not jumped to. (e.g. LABEL0 in this code sample fragment).

## Example

POU header
In the POU header, all input and output variables are declared that are used for programming this function.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | var_1 | INT | 0 | Input_1 |
| 1 | VAR | var_2 | INT | 0 | Input_2 |
| 2 | VAR | var_3 | INT | 0 | Input_3 |
| 3 | VAR | var_4 | INT | 0 | Input_4 |
| 4 | VAR | var_5 | INT | 0 | Output |

IL body

| | | | |
|---|---|---|---|
| | | | (* Beginning of network 0 *) |
| LABEL0: | LD | **var_1** | (* Load **var_1** in accu *) |

| | | | |
|---|---|---|---|
| | EQ | **var_2** | (* Compare accu with var_2; store result of comparison in accu; result is of type BOOL*) |
| | JMPC | LABEL1 | (* If accu contains FALSE continue program execution at position LABEL1 *) |
| | | | |
| | | | (* Beginning of network 1 *) |
| LABEL1: | MUL | **var_3** | (* Multiply accu by **var_3** *) |
| | SUB | **var_4** | (* Subtract **var_4** from accu; result is store in accu *) |
| | ST | **var_5** | (* Store accu in var_5 *) |

# RET

Unconditional Return from Function-Block

The **RET** command causes the return from a called function block to the main program.

Note

- Operator only available in IL programming language.
- This command takes no operand.

## Example

POU header
In the POU header, all input and output variables are declared that are used for programming this function.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | var_1 | INT | 0 | Input_1 |
| 1 | VAR | var_2 | INT | 0 | Input_2 |
| 2 | VAR | var_3 | INT | 0 | Input_3 |

IL body

| | | |
|---|---|---|
| | | (* In midst of some function block *) |
| LD | **var_1** | (* Load **var_1** in accu *) |
| MUL | **var_2** | (* Multiply accu by **var_2** *) |
| ST | **var_3** | (* Store accu in **var_3** *) |
| RET | | (* Continue execution with statement,<br>following the statement that called the function block *) |

# RETC

Conditional Return from Function-Block

If the content of the accumulator is TRUE, the **RETC** command causes the return from a called function block to the main program.

## Remarks

- The value in the accumulator must be of data type BOOL.
- Operator only available in IL programming language.
- This command takes no operand.

## Example

POU header
In the POU header, all input and output variables are declared that are used for programming this function.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | var_1 | INT | 0 | Input_1 |
| 1 | VAR | var_2 | INT | 0 | Input_2 |

IL body

| | | |
|---|---|---|
| | | (* In midst of some function block *) |
| LD | **var_1** | (* Load **var_1** in accu *) |
| EQ | **var_2** | (* **var_1** = **var_2**? *) |
| RETC | | (* If accu contains TRUE, continue execution with statement, following the statement that called the function block *) |

# RETCN

Conditional Return NOT from Function Block

If the content of the accumulator is FALSE, the **RETCN** command causes the return from a called function block to the main program.

## Remarks

- The value in the accumulator must be of data type BOOL.
- Operator only available in IL programming language.
- This command takes no operand.

## Example

### POU header

In the POU header, all input and output variables are declared that are used for programming this function.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | var_1 | INT | 0 | Input_1 |
| 1 | VAR | var_2 | INT | 0 | Input_2 |

### IL body

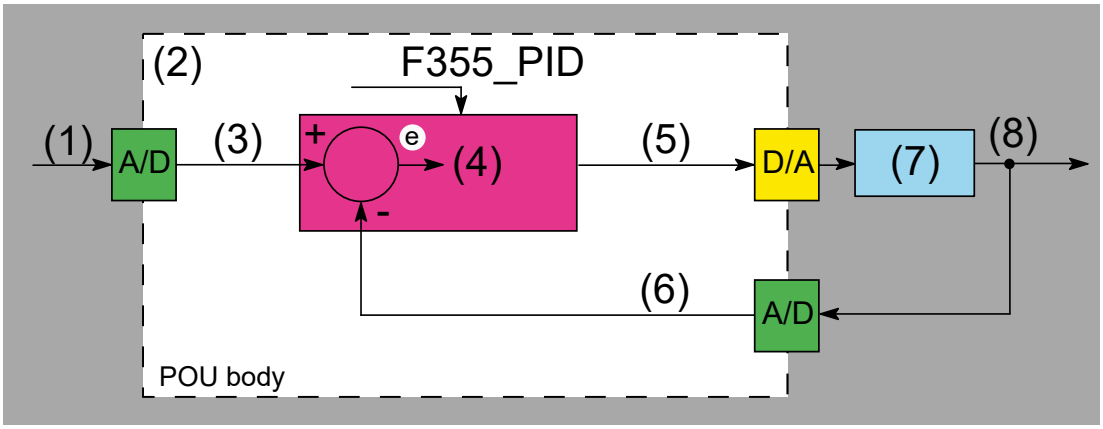
| | | |
|---|---|---|
| | | (* In midst of some function block *) |
| **LD** | **var_1** | (* Load **var_1** in accu *) |
| **EQ** | **var_2** | (* **var_1** = **var_2**? *) |
| **RETCN** | | (* If accu contains FALSE, continue execution with statement, following the statement that called the function block *) |

## 24.10 FP instructions

Tip

Advantages of FP instructions

### 24.10.1 Explanation of the operation of the PID instructions




(1) Control input
(2) Parameter (Kp, Ti, Td, Ts)
(3) Set point value (SP)
(4) PID calculation
(5) Manipulated variable (MV)
(6) Measured process value (PV)
(7) Analog section
(8) Output quantity

The above POU body represents the standard control loop. The control input is determined by the user (e.g. desired room temperature of 22°C). After the A/D conversion the set point value (SP) is entered as one input value for the PID processing instruction. The measured process value (PV) (e.g. current room temperature) is normally transmitted via a sensor and entered as another input value for the PID processor. **F355_PID** calculates the standard tolerance e (error value) from the set point value and the process value (e = set value - measured process value). With the parameters given (proportional gain Kp, integral time Ti, ...) a new output value (manipulated value MV) is calculated in increments set by the sampling time Ts. This result is then applied to the actuator (e.g. a fan that regulates room temperature) after the D/A conversion. The analog section represents the system's actuator, e.g. heater and temperature regulation of a room.

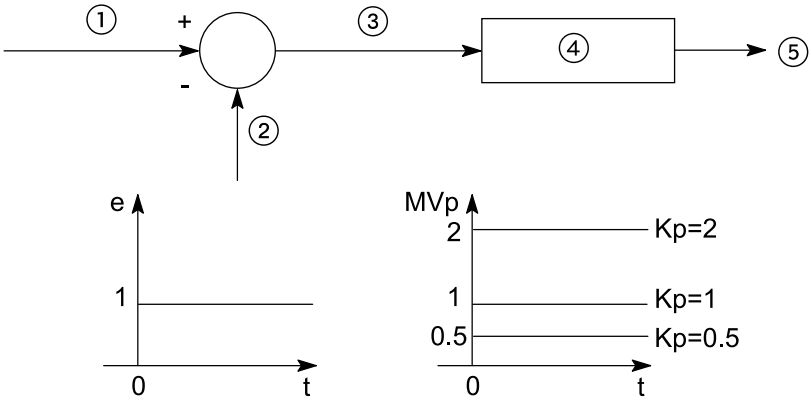
#### A PID operation consists of three components:

1. Proportional part (P part)

   A proportional part generates an output that is proportional to the error value. The proportional gain Kp determines by how much the manipulated value is increased or

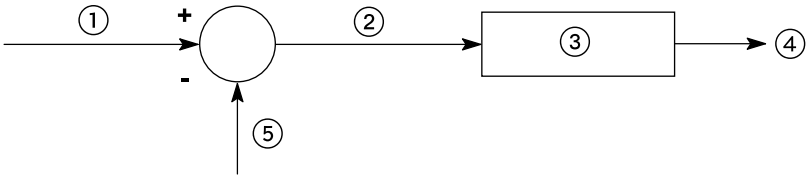
decreased. A proportional part can be a simple electric resistor or a linear amplifier, for example.

The P part displays a relatively large maximum overshot, a long settling time and a constant standard tolerance.



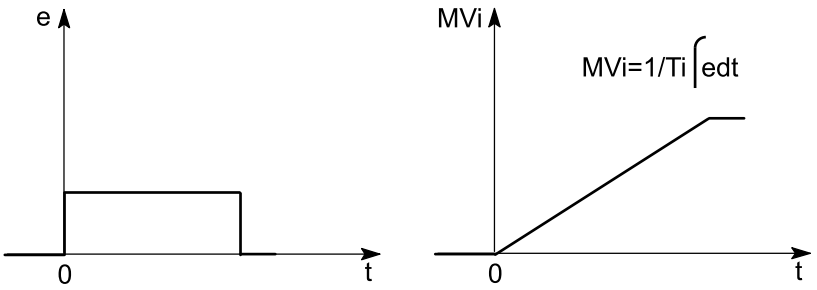

2. Integral part (I part)

   An integral part produces an output quantity that corresponds to the time integral and input quantity (area of the input quantity). The integral time thus evaluates the output quantity MVi.The integral part can be a quantity scale of a tank that is filled by a volume flow, for example. Because of the slow reaction time of the integral part, it has a larger maximum overshot than the P component, but no remaining constant error value.



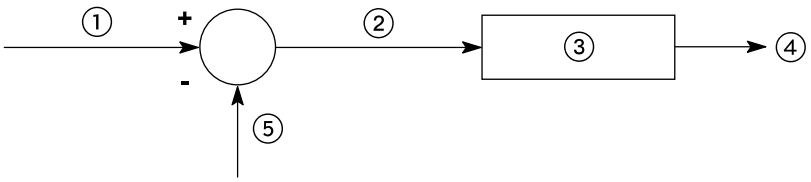

   Example:

   Input quantity **e** and the output quantity **MVi** produced.



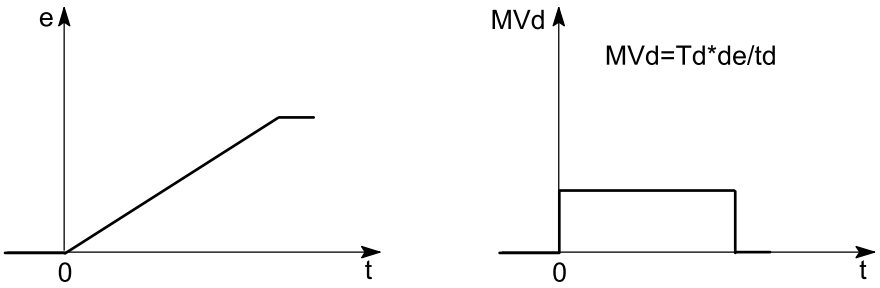

3. Derivative part (D part)

   The derivative part produces an output quantity that corresponds to the time derivation of the error value. The derivative time corresponds to the weighting of the derived input quantity. A derivative component can be an RC-bleeder (capacitor hooked up in series and resistance in parallel), for example.



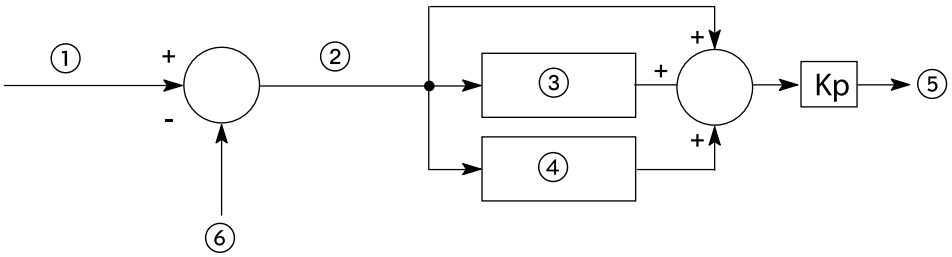

Example:

Input quantity **e** and the output quantity **MVd** produced.




4. PID controller

   A PID controller is a combination of a P component, an I component and a D component. When the parameters Kp, Ti and Td are optimally adjusted, a PID controller can quickly control and maintain a quantity at a predetermined set value.




## Reference equations for calculating the controller output MV

The following equations are used to calculate the controller output MV under the following conditions:

In general:The output value at time period **n** is calculated from the previous output value (n-1) and the change in the output value in this time interval.

$$MV_n = MV_{n-1} + \Delta MV$$

- Reverse operation PI-D Control = 16#X000

$$\Delta MV = Kp \times \left[ (e_n - e_{n-1}) + e_n \times \frac{Ts}{Ti} + \Delta D_n \right]$$

$$e_n = SP_n - PV_n$$

$$\Delta Dn = (\eta\beta - 1)D_{n-1} + \beta(PV_{n-1} - PV_n)$$

$$\eta = \frac{1}{8}(\text{constant})$$

$$\beta = \frac{Td}{(Ts + \eta Td)}$$

- Forward operation PI-D Control = 16#X001

$$\Delta MV = Kp \times \left[ (e_n - e_{n-1}) + e_n \times \frac{Ts}{Ti} + \Delta D_n \right]$$

$$e_n = PV_n - SP_n$$

$$\Delta Dn = (\eta\beta - 1)D_{n-1} + \beta(PV_n - PV_{n-1})$$

$$\eta = \frac{1}{8}(\text{constant})$$

$$\beta = \frac{Td}{(Ts + \eta Td)}$$

- Reverse operation I-PD Control = 16#X002

$$\Delta MV = Kp \times \left[ (PV_{n-1} - PV_n) + e_n \times \frac{Ts}{Ti} + \Delta D_n \right]$$

$$e_n = SP_n - PV_n$$

$$\Delta Dn = (\eta\beta - 1)D_{n-1} + \beta(PV_{n-1} - PV_n)$$

$$\eta = \frac{1}{8}(\text{constant})$$

$$\beta = \frac{Td}{(Ts + \eta Td)}$$

- Forward operation I-PD Control = 16#X003

$$\Delta MV = Kp \times \left[ (PV_n - PV_{n-1}) + e_n \times \frac{Ts}{Ti} + \Delta D_n \right]$$

$$e_n = PV_n - SP_n$$

$$\Delta Dn = (\eta\beta - 1)D_{n-1} + \beta(PV_n - PV_{n-1})$$

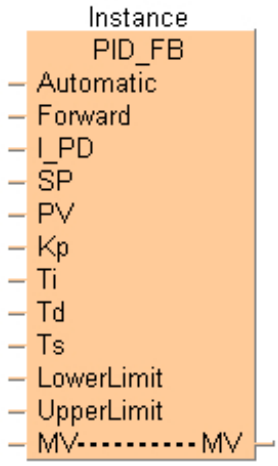
$$\eta = \frac{1}{8}(\text{constant})$$

$$\beta = \frac{Td}{(Ts + \eta Td)}$$

# PID_FB

PID processing instruction

This implementation allows you to set the parameters of **F355_PID** directly using arguments:




## Parameters

Input

**Automatic**

FALSE: Manual setting of MV possible
TRUE: Automatic PID controlled MV

**Forward** (BOOL)

FALSE: Inverse control (heating)
TRUE: Forward control (cooling)

**I_PD** (BOOL)

FALSE: PI-D control
TRUE: I-PD control

**SP** (INT)

Set point value, range 0-10000

**PV** (INT)

Process value, range 0-10000

**Kp** (INT)

Proportional gain, range: 1-9999, unit: 0.1

**Ti** (INT)

Integral time, range: 1-30000, unit: 0.1s

**Td** (INT)

Derivative time, range: 1-10000, unit: 0.1s

**Ts** (INT)

Sampling time, range: 1-6000, unit: 0.01s

**LowerLimit** (INT)

MV lower limit, range: 0-10000

**UpperLimit** (INT)

MV upper limit, range: 1-10000

Input/output

**MV** (INT)

Manipulated value

## Remarks

- Auto-tuning is not possible using **PID_FB**. For this, use **PID_FB_DUT**.
- The value for **MV** can be assigned externally either when the program is initialized or when the value of **Automatic** is FALSE.
- In order to achieve maximum resolution and minimum dead time beyond **LowerLimit** and **UpperLimit**, their values should, if possible, cover the entire range of 0–10000.

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

if the parameter settings are outside the permissible range

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

if the parameter settings are outside the permissible range

## Example

## Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global Variables

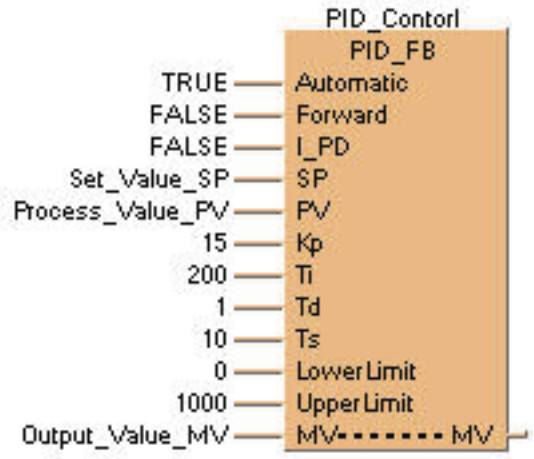
| | Class | Identifier | FP ... | IEC Address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | Set_Value_SP | | | INT | 0 |
| 1 | VAR_GLOBAL | Process_V... | | | INT | 0 |
| 2 | VAR_GLOBAL | Output_Val... | | | INT | 0 |

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | Set_Value_SP | INT | 0 |
| 1 | VAR_EXTERNAL | Process_Value_PV | INT | 0 |
| 2 | VAR_EXTERNAL | Output_Value_MV | INT | 0 |
| 3 | VAR | PID_Control | PID_FB | |

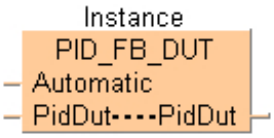
## LD body



# PID_FB_DUT

PID processing instruction

This implementation allows you to access the **F355_PID** instruction via the structure **PID_DUT_31**.




## Parameters

Input

**Automatic** (BOOL)

FALSE: Manual setting of MV possible
TRUE: Automatic PID controlled MV

Output

**PidDut** (PID_DUT_31)

For a detailed explanation of parameters, please refer to **PID_DUT_31**

## Remarks

- The value for **MV** can be assigned externally either when the program is initialized or when the value of **Automatic** is FALSE.
- In order to achieve maximum resolution and minimum dead time beyond **LowerLimit** and **UpperLimit**, these values should, if possible, cover the entire range of 0–10000.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if the parameter settings are outside the permissible range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if the parameter settings are outside the permissible range

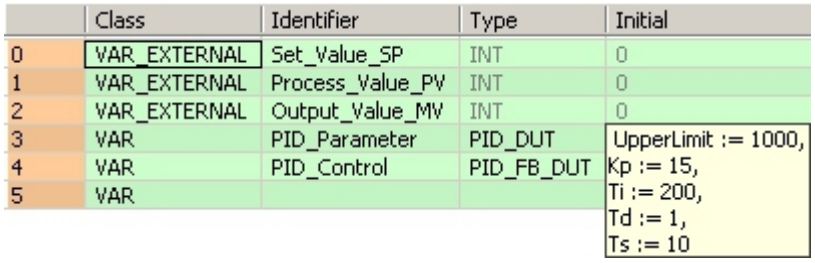
## Example

### Global variables

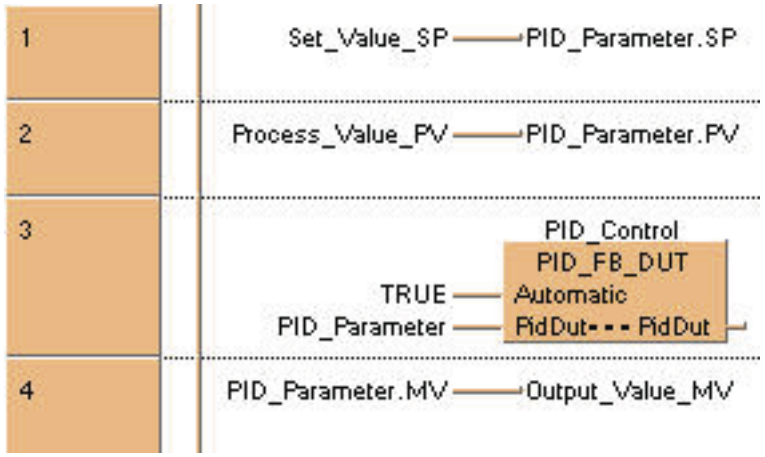
In the global variable list you define variables that can be accessed by all POUs in the project.


### POU header

All input and output variables used for programming this function have been declared in the POU header.  The same POU header is used for all programming languages.



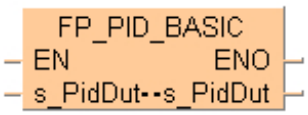

### LD body



# FP_PID_BASIC

PID processing

This FP instruction regulates a process (e.g. a heater) given a measured value (e.g. temperature) and a predetermined output value (e.g. 20°C).




## Parameters

Input/output

**s_PidDut** (PID_DUT_31)

Detailed explanation of parameters, please refer to **PID_DUT_31**

## Remarks

The function calculates a PID algorithm whose parameters are determined in a data table in the form of an ARRAY with 30 elements that is entered at input **s_PidDut**.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if the parameter settings are outside the permissible range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if the parameter settings are outside the permissible range

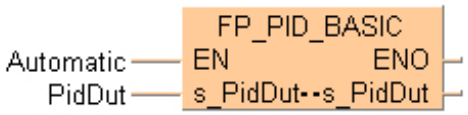
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Automatic | BOOL | TRUE |
| 1 | VAR | PidDut | PID_DUT_31 | |

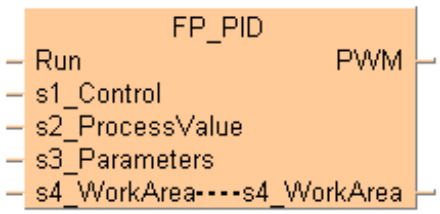
### LD body



# FP_PID

PID processing with optional PWM output

PID processing is performed to keep the process value PV as close as possible to the set point value SP. In contrast to **FP_PID_BASIC**, this instruction enables a **PWM** output (on-off output). Auto-tuning is also available to automatically calculate the PID control data Kp, Ti, and Td.




## Parameters

Input

**Run** (BOOL)

Start condition

**s1_Control** (**PID_Control_DUT**)

Control data

**s2_ProcessValue** (INT)

Process value (-30000–30000)

**s3_Parameters** (**PID_PARAMETERS_DUT**)

PID control parameters

Input/output

**s4_WorkArea**( **PID_WORK_AREA_DUT**)

Data table of PID parameters

### Note

When you execute the instruction for the first time, i.e. when the execution condition specified at **Run** turns to TRUE, the default values are written to the DUT members 1 to 9 of DUT .
Before the second execution of **FP_PID**, you need to change the DUT members 1 to 9 of the DUT to the required values.

Output

**PWM** (BOOL)

Pulse-width modulated output (optional, can be used instead of manipulated value output)

## Remarks

- The period (cycle) of the **PWM** output is the sampling time Ts (the frequency of the **PWM** output is 1/Ts) and the duty is the manipulated value MV in 0.01% units, e.g. MV = 10000 means a duty of 100%.
- The instruction has to be executed twice. The first execution resets the values in **PID_WORK_AREA_DUT**, then the correct values for the DUT members 1 to 9 in **PID_WORK_AREA_DUT** need to be set so that the second execution of the instruction uses the correct values.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter of **PID_Parameters_DUT** is outside the permissible range

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter of **PID_Parameters_DUT** is outside the permissible range

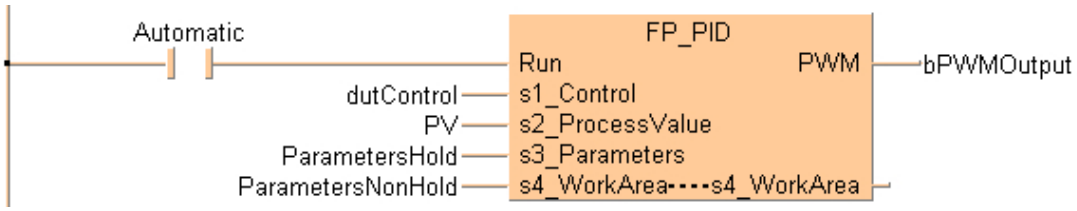
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Automatic | BOOL | TRUE |
| 1 | VAR | dutControl | PID_Control_DUT | |
| 2 | VAR | PV | INT | 0 |
| 3 | VAR | ParametersHold | PID_Parameters_DUT | |
| 4 | VAR | ParametersNonHold | PID_Work_Area_DUT | |
| 5 | VAR | bPWMOutput | BOOL | FALSE |

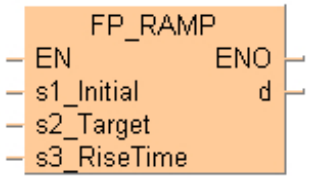
### LD body


Automatic
FP_PID
Run
PWM
bPWMOutput
dutControl
s1_Control
PV
s2_ProcessValue
ParametersHold
s3_Parameters
ParametersNonHold
s4_WorkArea
s4_WorkArea

# FP_RAMP

Ramp output

This FP instruction calculates a linear ramp based on an initial value, a target value and a time range if the trigger **EN** is TRUE. The output value **d** increases or decreases until the target value is reached.




## Parameters

Input

**s1_Initial** (INT, DINT, UINT, UDINT, REAL, LREAL)

Initial value from which the output value increases or decreases.

**s2_Target** (INT, DINT, UINT, UDINT, REAL, LREAL)

Target value to which the output value increases or decreases.

**s3_RiseTime** (INT, DINT, UINT, UDINT)

Time range in ms for the output value to increase or decrease from the initial value to the target value.
Values: 1–30000

Output

**d** (INT, DINT, UINT, UDINT, REAL, LREAL)

Output value

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the output time range specified by **s3_RiseTime** is smaller than 1 or larger than 30000.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.

- if the output time range specified by **s3_RiseTime** is smaller than 1 or larger than 30000.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

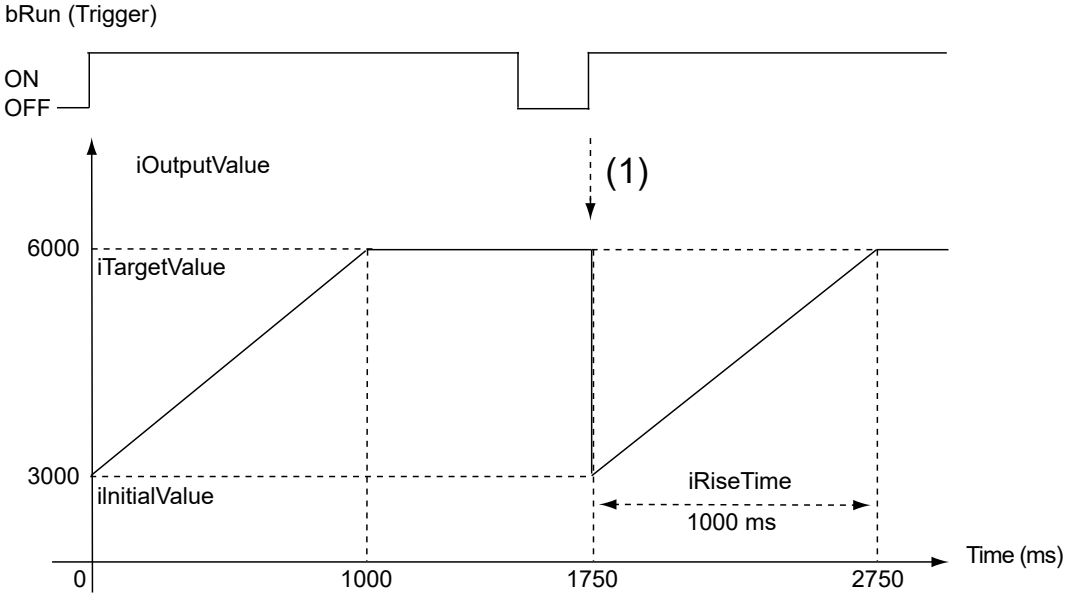
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iInitialValue | INT | 3000 |
| 1 | VAR | iTargetValue | INT | 6000 |
| 2 | VAR | iRiseTime | INT | 1000 |
| 3 | VAR | iOutputValue | INT | 0 |
| 4 | VAR | bRun | BOOL | FALSE |

### POU body

When the variable **bRun** is switched to TRUE, the function is carried out and **iOutputValue** increases from 3000 (**iInitialValue**) to 6000 (**iTargetValue**) in 1000ms (according to **iRiseTime**).

- Time chart for increasing the output value:

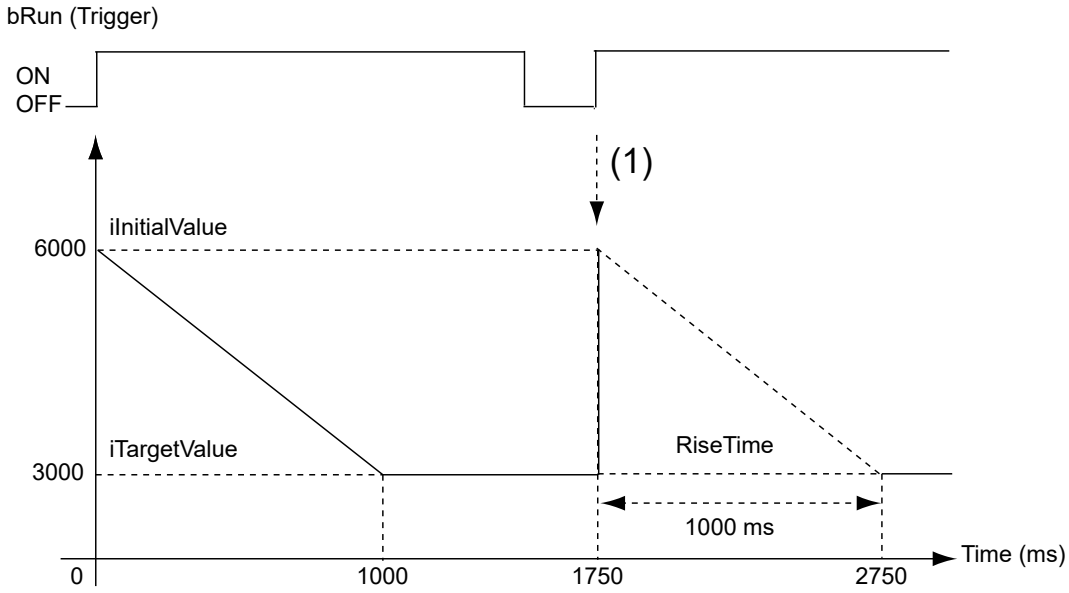
  Example values: **iInitialValue** = 3000, **iTargetValue** = 6000, **iRiseTime** = 1000




  (1) Data is initialized when the system detects the trigger's rising edge.

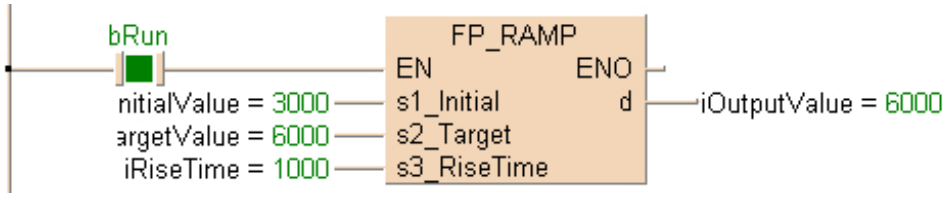
- Time chart for decreasing the output value:

  Example values: **iInitialValue** = 6000, **iTargetValue** = 3000, **iRiseTime** = 1000




(1) Data is initialized when the system detects the trigger's rising edge.

## LD body



## 24.11 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

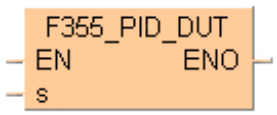
Advantages of IEC instructions

Advantages of FP instructions

# F355_PID_DUT

PID processing instruction

The PID processing instruction is used to regulate a process (e.g. a heater) given a measured value (e.g. temperature) and a predetermined output value (e.g. 20°C).




## Parameters

Input

**s** (PID_DUT_31)

Detailed explanation of parameters, please refer to **PID_DUT_31**

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction: FP_PID_BASIC
- The function calculates a PID algorithm whose parameters are determined in a data table in the form of an ARRAY with 30 elements that is entered at input **s**.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if the parameter settings are outside the permissible range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if the parameter settings are outside the permissible range

## Example

## Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global Variables

| | Class | Identifier | FP ... | IEC_Address | Type | Initial | A... | Comment |
|---|---|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | EnableAutoTuning | X0 | %IX0.0 | BOOL | FALSE | | Switch Autotuning on |
| 1 | VAR_GLOBAL | Set_Value_SP | WX4 | %IW4 | INT | 0 | | A/D CH0 |
| 2 | VAR_GLOBAL | Process_Value_PV | WX5 | %IW5 | INT | 0 | | A/D CH1 |
| 3 | VAR_GLOBAL | Output_Value_MV | WY4 | %QW4 | INT | 0 | | D/A |

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

PID_Control_IL

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | EnableAutoTuning | BOOL | FALSE | Switch Autotuning on |
| 1 | VAR_EXTERNAL | Set_Value_SP | INT | 0 | A/D CH0 |
| 2 | VAR_EXTERNAL | Process_Value_PV | INT | 0 | A/D CH1 |
| 3 | VAR_EXTERNAL | Output_Value_MV | INT | 0 | D/A |
| 4 | VAR | PidParameters | PID_DUT_31 | | PID Parameters |

In the initialization of the variable **PidParameters** of the data type **PID_DUT_31**, the **MV** upper limit is set to 4000. The proportional gain **Kp** is initially set at 80 (8), **Ti** and **Td** at 200 (20s) and the sampling time **Ts** at 100 (1s).

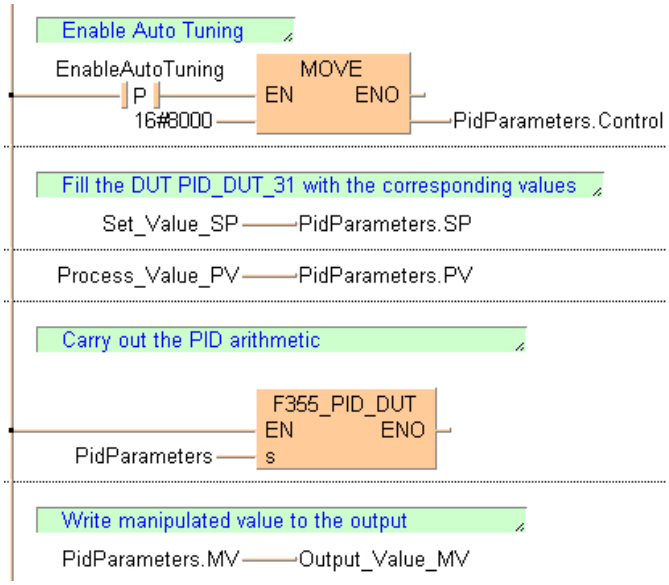
## POU body

The standard function **MOVE** copies the value 16#8000 to the member **Control** of the DUT **PidParameters** when the variable **EnableAutoTuning** turns from FALSE to TRUE (i.e. activates the control mode auto-tuning in the function **F355_PID_DUT**).

The variables **Set_Value_SP** and **Process_Value_PV** are assigned to the members **SP** and **PV** of the DUT **PidParameters**. They receive their values from the A/D converter channel 0 and 1.

Because the **F355_PID_DUT** function block has an EN output connected directly to the power rail, the function is carried out when the PLC is in RUN mode. The calculated controller output stored by the member **MV** of the DUT **PidParameters** is assigned to the variable **Output_Value_MV**. Its value is returned via a D/A converter from the PLC to the output of the system.

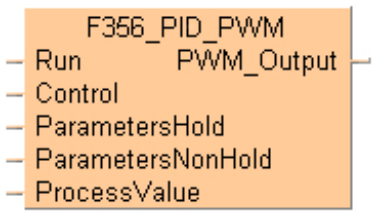
### LD body


Enable Auto Tuning
EnableAutoTuning
P
MOVE
EN
ENO
16#8000
PidParameters.Control
Fill the DUT PID_DUT_31 with the corresponding values
Set_Value_SP
PidParameters.SP
Process_Value_PV
PidParameters.PV
Carry out the PID arithmetic
F355_PID_DUT
EN
ENO
PidParameters
s
Write manipulated value to the output
PidParameters.MV
Output_Value_MV

# F356_PID_PWM

PID processing with optional PWM output

PID processing is performed to keep the process value PV as close as possible to the set point value SP. In contrast to **F355_PID_DUT**, this instruction enables a PWM output (on-off output). Auto-tuning is also available to automatically calculate the PID control data Kp, Ti, and Td.




## Parameters

Input

**Run** (BOOL)

Start condition

**Control** (F356_Control_DUT)

Control data

**ParametersHold** (F356_Parameters_Hold_DUT)

PID control parameters

**ParametersNonHold** (F356_Parameters_NonHold_DUT)

Manipulated value MV, additional control mode area, auto-tuning related area and working area

Note

When you execute the instruction for the first time, i.e. when the execution condition specified at **Run** turns to TRUE, the default values are written to the DUT members 1 to 9 of DUT .

**ProcessValue** (INT)

Process value (-30000–30000)

Output

**PWM_Output** (BOOL) (see note)

Pulse-width modulated output (optional, instead of manipulated value output)

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_PID**
- Abbreviations used when describing PID processing

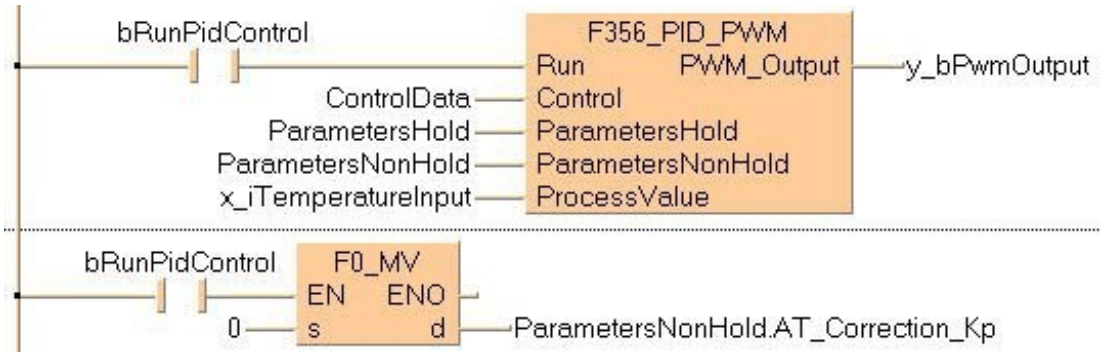
| Abbreviation | What it stands for | Also know as |
|---|---|---|
| **PV** | Process value | Actual value, measured value |
| **SP** | Set point value | Target value, set value |
| **MV** | Manipulated value | Output value, manipulated variable |
| **Ts** | Sampling time | Cycle time |
| **Ti** | Integral time | - |
| **Td** | Derivative time | - |
| **Kp** | Proportional gain | - |
| **AT** | Auto-tuning | - |

- Before the second execution of **FP_PID**, you need to change the DUT members 1 to 9 of the DUT to the required values.

## General programming information

1. When the input at **Run** is executed, the data in the argument **ParametersNonHold** is initialized. If you want a value in the DUT to use non-default values, write the values into the DUT using a **MOVE** instruction, for example, which must be triggered continuously by a TRUE condition.
2. **F356_PID_PWM** must be executed once and only once per scan. Therefore, do not execute **F356_PID_PWM** in interrupt programs or loops.
3. Do not turn the execution condition to FALSE during PID processing. Otherwise, PID processing will be disabled.
4. If you do not want parallel PWM output cycles, e.g. to enable control of multiple objects, delay the start-up times accordingly, e.g. by employing a timer instruction.

### Example




Note

The period (cycle) of the PWM output is the sampling time Ts (the frequency of the PWM output is 1/Ts) and the duty is the manipulated value MV in 0.01% units, e.g. MV = 10000 means a duty of 100%.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if any parameter of **F356_Parameters_NonHold_DUT** is out of range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if any parameter of **F356_Parameters_NonHold_DUT** is out of range

**sys_bIsEqual** (turns to TRUE and remains TRUE)

if the area specified with **UpperLimit** or **LowerLimit** is out of range

## Detailed information:

- Control conditions: **F356_Parameters_Hold_DUT**
- Set point value SP and the control parameters: **F356_Parameters_Hold_DUT**

Additional notes on auto-tuning:

- The members **AT_Progress** in **F356_Parameters_Hold_DUT** and **b1_AT_Complete** in **F356_Control_DUT** are cleared at the rising edge of the auto-tuning signal.
- When auto-tuning has completed successfully, the element **b1_AT_Complete** of **F356_Control_DUT** is set, and the auto-tuning done code is stored in the element **AT_Progress** of **F356_Parameters_NonHold_DUT**.
- When auto-tuning is aborted, the parameters of Kp, Ti, and Td are not changed.

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global Variables

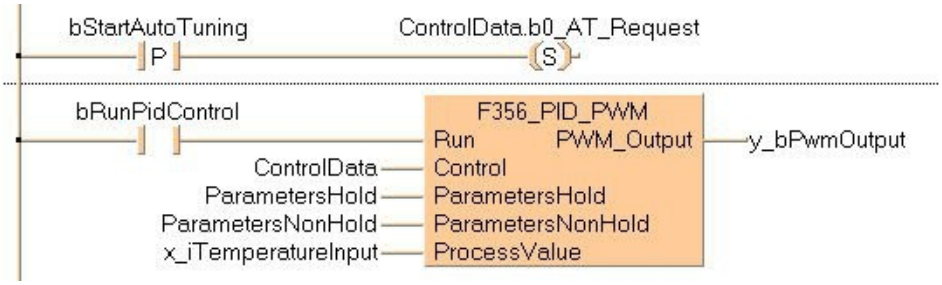
| | Class | Identifier | FP Address | IEC Address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | x_iTemperatureInput | WX2 | %IW2 | INT | 0 |
| 1 | VAR_GLOBAL | y_bPwmOutput | Y0 | %QX0.0 | BOOL | FALSE |
| 2 | VAR_GLOBAL | | | | | |

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | x_iTemperatureInput | INT | 0 |
| 1 | VAR_EXTERNAL | y_bPwmOutput | BOOL | FALSE |
| 2 | VAR | bStartAutoTuning | BOOL | FALSE |
| 3 | VAR | bRunPidControl | BOOL | FALSE |
| 4 | VAR | ControlData | F356_Control_DUT | |
| 5 | VAR | ParametersHold | F356_Parameters_Hold_DUT | |
| 6 | VAR | ParametersNonHold | F356_Parameters_NonHold_DUT | |

### LD body



# 25 Program execution control instructions

# BRK

Break

The **BRK** (Breakpoint) instruction stops the execution at the address of this instruction during the test run mode if the trigger **EN** is in the ON-state.


## Remarks

- Once this instruction is executed, the program halts. To continue the program, the mode in the test run (continuous run / step run) should be selected. In the step run mode, the program is executed instruction by instruction regardless of the instructions and in the continuous run mode, the program is executed until it is stopped by the next break instruction (**BRK**) or the end of the program (end instruction ED).
- The test run mode is executed, when the mode selector switch on the PLC is set to RUN mode with setting the INITIALIZE/TEST switch to the TEST mode.

## Example

### POU header

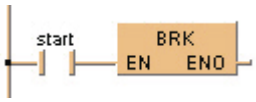
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

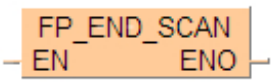
### LD body



# FP_END_SCAN

Conditional end

Ends the current scan of the program code at the current program position.




## Parameters

Input

**EN** (BOOL)

if TRUE, the instruction is carried out

Output

**ENO** (BOOL)

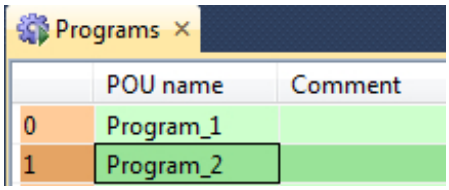
set to TRUE, if the instruction has been successfully processed

## Remarks

- When the execution condition turns to TRUE, the program ends the current scan and begins I/O processing. Then, the program returns to the first address.
- This instruction cannot be performed in sub-programs such as subroutine programs or interrupt programs. Use the **FP_END_SCAN** instruction in the main program area only.
- Two or more **FP_END_SCAN** instructions can be used within the main program.
- You must be careful when using one of the instructions listed below, which are executed by detecting the rising edge of a execution condition (trigger).
  - **DF** (rising edge differential)
  - Count input for **CT** (counter)
  - Count input for **F118_UDC** (up/down counter)
  - Shift input for SR (shift register)
  - Shift input for **F119_LRSR** (left/right shift register)
  - Differential execution of P instructions (specified by P and a number, e.g. P20_ADD)

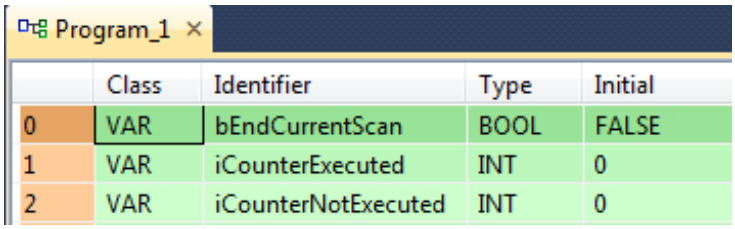
## Example

### Task pool

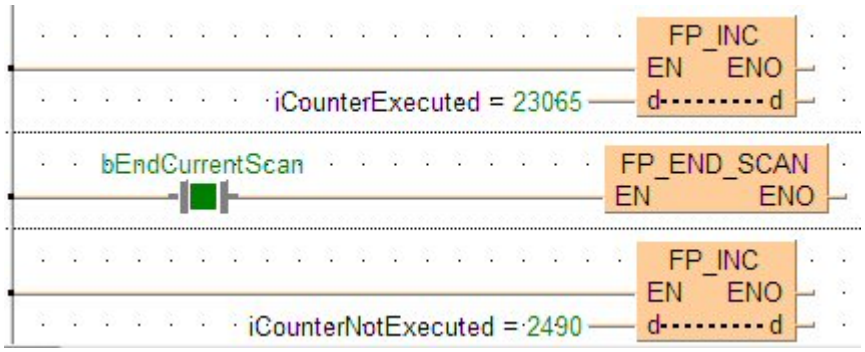



### POU header

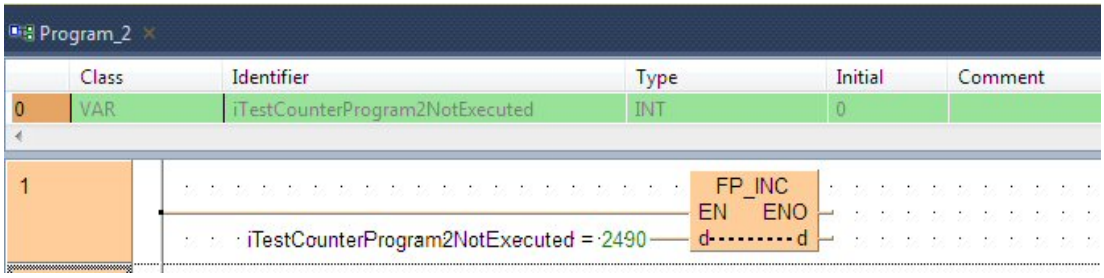
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.




### LD body




### Example Program_2

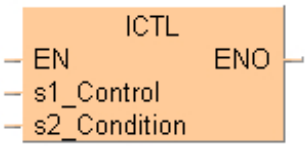


# ICTL

Interrupt control

The **ICTL** instruction sets all interrupts to enable or disable. Each time the **ICTL** instruction is executed, it is possible to set parameters like the type and validity of interrupt programs. Settings can be specified by **s1_Control** and **s2_Condition**.

- **s1_Control**16-bit equivalent constant or 16-bit area for interrupt control setting
- **s2_Condition**16-bit equivalent constant or 16-bit area for interrupt trigger condition setting




## Parameters

Input

**s1_Control** (WORD, INT, UINT)

Interrupt control data setting

**s2_Condition** (WORD, INT, UINT)

Interrupt condition setting

## Remarks

The number of interrupt programs available is:

| | |
|---|---|
| 16 | interrupt module initiated interrupt programs (INT 0–INT 15) |
| 8 | advanced module (special modules, like positioning,...) initiated interrupt programs (INT 16–INT 23) |
| 1 | periodic interrupt program (INT 24) (Time base 0.5ms selectable for FP2/2SH, FP10SH) |

Be sure to use **ICTL** instructions so that they are executed once at the rising edge of the **ICTL** trigger using the **DF** (page 1132) instruction.

Two or more **ICTL** instructions can have the same trigger.

| Bit | 15 .. 8 | 7 .. 0 |
|---|---|---|
| **s1_Control**<br>**16#** | **Selection of control function**<br>**00**: Interrupt "enable/disable" control<br>**01**: Interrupt trigger reset control | **Interrupt type selection**<br>**00**: Interrupt module (INT 0–15)<br>**01**: Advanced module (INT 16–23)<br>**02**: Periodic interrupt (INT 24) |
| **s2_Condition**<br>**2#** | Bit 0: **0** Interrupt program 0 disabled<br>Bit 0: **1** Interrupt program 0 enabled<br>Bit 1: **0** Interrupt program 1 disabled<br>...<br>Bit 15: **1** Interrupt program 15 enabled<br>Example: **s2** = 2#0000000000001010 | |

Note

- The current enable/disable status of each interrupt module initiated interrupt can be checked by monitoring the system variable **sys_wInterruptMask_0_15**.
- The current enable/disable status of each non-interrupt module initiated interrupt can be checked by monitoring the system variable **sys_wInterruptMask_16_31**.
- The current interrupt interval of the periodic interrupt can be checked by monitoring the system variable **sys_iPeriodicInterruptInterval**.
- If a program is written into an interrupt task, the interrupt concerned will be enabled automatically during the initialization routine when starting the program.
- With the **ICTL** instruction an interrupt task can be enabled or disabled by the program.

## Example

### POU header

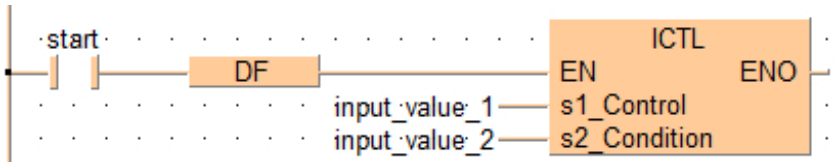
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 1 | VAR | start | BOOL | FALSE | enable signal |
| 2 | VAR | input_value_1 | WORD | 16#0002 | first input parameter |
| 3 | VAR | input_value_2 | WORD | 10 | second input parameter |

### POU body

The interval for executing the periodic interrupt is specified as 100ms (10ms time base selected) when the rising edge of start is detected.

### LD body


start
DF
ICTL
EN
ENO
input_value_1
s1_Control
input_value_2
s2_Condition

## 25.4 FP instructions

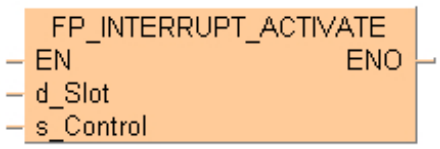
Tip

Advantages of FP instructions

# FP_INTERRUPT_ACTIVATE

Control the activation of interrupt programs for one unit

This FP instruction controls the activation of interrupt programs for the unit installed in the slot specified by **d_Slot** according to the data specified by **s_Control**.




## Parameters

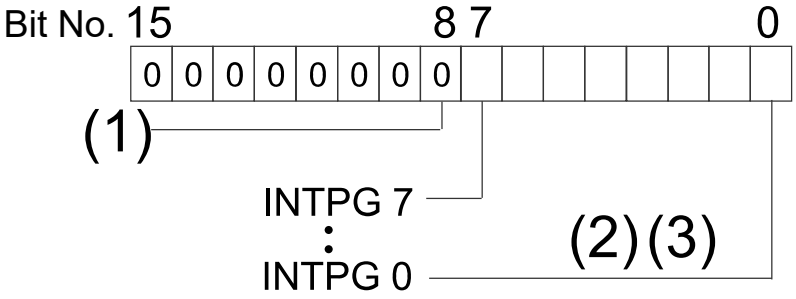
Input

**d_Slot** (WORD, INT, UINT)

Slot number: 1–16

**s_Control** (WORD)

Control word to activate the interrupt number of the unit installed in the specified slot: 16#0–FF




(1) Higher 8 bits: fixed at 0
(2) 0: Interrupt disabled
(3) 1: Interrupt enabled

## Remarks

- The interrupt program (INTPG 0-7) which should be executed for a given interrupt number can be assigned to the interrupt task with the appropriate number.
- The interrupt number can be calculated using the slot number as follows: <slot number> * 10 + INTPG.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if the area specified using the index modifier exceeds the limit.

## Example

## Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

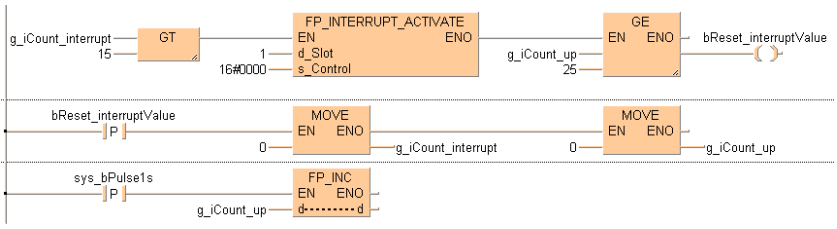
| | Class | Identifier | FP address | IEC address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | g_iCount_interrupt | | | INT | 0 |
| 1 | VAR_GLOBAL | g_iCount_up | | | INT | 0 |
| 2 | VAR_GLOBAL | g_bHscUnitEnable | Y0 | %QX0.0 | BOOL | FALSE |
| 3 | VAR_GLOBAL | g_bHscUnitEnableCounting | Y1 | %QX0.1 | BOOL | FALSE |
| 4 | VAR_GLOBAL | g_bIsHscUnitEnabled | X0 | %IX0.0 | BOOL | FALSE |
| 5 | VAR_GLOBAL | g_bIsHscUnitCounting | X1 | %IX0.1 | BOOL | FALSE |
| 6 | VAR_GLOBAL | g_bHscUnitComparisonMatchFlag | X10 | %IX1.0 | BOOL | FALSE |
| 7 | VAR_GLOBAL_CONSTANT | iACTIVATE_1_0 | | | INT | 0 |

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_iCount_interrupt | INT | 0 |
| 1 | VAR_EXTERNAL | g_bHscUnitEnable | BOOL | FALSE |
| 2 | VAR_EXTERNAL | g_bHscUnitEnableCounting | BOOL | FALSE |
| 3 | VAR_EXTERNAL | g_bHscUnitComparisonMatchFlag | BOOL | FALSE |
| 4 | VAR_EXTERNAL | g_iCount_up | INT | 0 |
| 5 | VAR | bEnable_Interrupt | BOOL | FALSE |
| 6 | VAR | bComparisonMatchMonitor | BOOL | FALSE |
| 7 | VAR | bDisableHscUnit | BOOL | FALSE |
| 8 | VAR | bDisable_Interrupt | BOOL | FALSE |
| 9 | VAR | bReset_interruptValue | BOOL | FALSE |

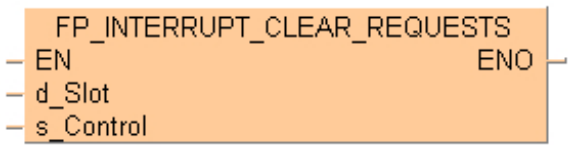
## LD body



# FP_INTERRUPT_CLEAR_REQUESTS

Clear interrupt programs for one unit

This FP instruction clears interrupt programs for the unit installed in the slot specified by **d_Slot** according to the data specified by **s_Control**.




## Parameters

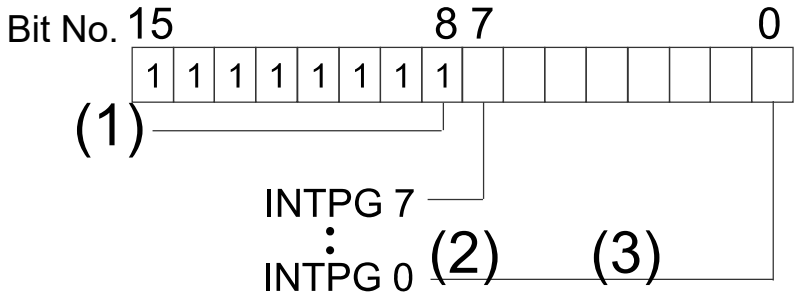
Input

**d_Slot** (WORD, INT, UINT)

Slot number: 1–16

**s_Control** (WORD)

Control word to clear the interrupt number of the unit installed in the specified slot: 16#0–FF




(1) Higher 8 bits: fixed at 1
(2) 0: Interrupt cleared
(3) 1: Interrupt not cleared

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if the area specified using the index modifier exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if the area specified using the index modifier exceeds the limit.

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global variables

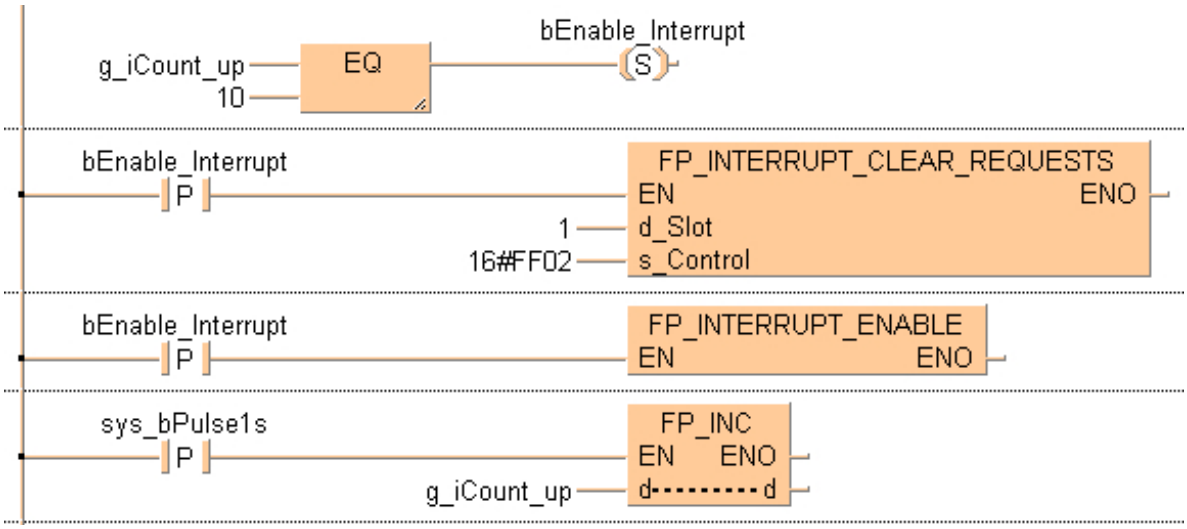
| | Class | Identifier | FP a... | IEC address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | g_bHscUnitComparisonMatchFlag_Ch0_0 | X10 | %IX1.0 | BOOL | FALSE |
| 1 | VAR_GLOBAL | g_bHscUnitComparisonMatchFlag_Ch0_1 | X11 | %IX1.1 | BOOL | FALSE |
| 2 | VAR_GLOBAL | g_bHscUnitComparisonMatchFlag_Ch1_0 | X30 | %IX3.0 | BOOL | FALSE |
| 3 | VAR_GLOBAL | g_bHscUnitComparisonMatchFlag_Ch1_1 | X31 | %IX3.1 | BOOL | FALSE |
| 4 | VAR_GLOBAL | g_bHscUnitEnable_Ch0 | Y0 | %QX0.0 | BOOL | FALSE |
| 5 | VAR_GLOBAL | g_bHscUnitEnable_Ch1 | Y10 | %QX1.0 | BOOL | FALSE |
| 6 | VAR_GLOBAL | g_bHscUnitEnableCounting_Ch0 | Y1 | %QX0.1 | BOOL | FALSE |
| 7 | VAR_GLOBAL | g_bHscUnitEnableCounting_Ch1 | Y11 | %QX1.1 | BOOL | FALSE |
| 8 | VAR_GLOBAL | g_iCount_I1_0 | | | INT | 0 |
| 9 | VAR_GLOBAL | g_iCount_I1_1 | | | INT | 0 |
| 10 | VAR_GLOBAL | g_iCount_I1_2 | | | INT | 0 |
| 11 | VAR_GLOBAL | g_iCount_I1_3 | | | INT | 0 |
| 12 | VAR_GLOBAL | g_iCount_up | | | INT | 0 |

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_bHscUnitComparisonMatchFlag_Ch0_0 | BOOL | FALSE |
| 1 | VAR_EXTERNAL | g_bHscUnitComparisonMatchFlag_Ch0_1 | BOOL | FALSE |
| 2 | VAR_EXTERNAL | g_bHscUnitComparisonMatchFlag_Ch1_0 | BOOL | FALSE |
| 3 | VAR_EXTERNAL | g_bHscUnitComparisonMatchFlag_Ch1_1 | BOOL | FALSE |
| 4 | VAR_EXTERNAL | g_bHscUnitEnable_Ch0 | BOOL | FALSE |
| 5 | VAR_EXTERNAL | g_bHscUnitEnable_Ch1 | BOOL | FALSE |
| 6 | VAR_EXTERNAL | g_bHscUnitEnableCounting_Ch0 | BOOL | FALSE |
| 7 | VAR_EXTERNAL | g_bHscUnitEnableCounting_Ch1 | BOOL | FALSE |
| 8 | VAR_EXTERNAL | g_iCount_I1_0 | INT | 0 |
| 9 | VAR_EXTERNAL | g_iCount_I1_1 | INT | 0 |
| 10 | VAR_EXTERNAL | g_iCount_I1_2 | INT | 0 |
| 11 | VAR_EXTERNAL | g_iCount_I1_3 | INT | 0 |
| 12 | VAR_EXTERNAL | g_iCount_up | INT | 0 |
| 13 | VAR | bComparisonMatchMonitor_0 | BOOL | FALSE |
| 14 | VAR | bComparisonMatchMonitor_1 | BOOL | FALSE |
| 15 | VAR | bComparisonMatchMonitor_2 | BOOL | FALSE |
| 16 | VAR | bComparisonMatchMonitor_3 | BOOL | FALSE |
| 17 | VAR | bDisable_Interrupt | BOOL | FALSE |
| 18 | VAR | bDisableHscUnit | BOOL | FALSE |
| 19 | VAR | bEnable_Interrupt | BOOL | FALSE |

### LD body


g_iCount_up
10
EQ
bEnable_Interrupt
S
bEnable_Interrupt
P
FP_INTERRUPT_CLEAR_REQUESTS
EN
ENO
1
d_Slot
16#FF02
s_Control
bEnable_Interrupt
P
FP_INTERRUPT_ENABLE
EN
ENO
sys_bPulse1s
P
FP_INC
EN
ENO
g_iCount_up
d
d

# FP_INTERRUPT_DISABLE

Disable interrupt programs

This FP instruction disables the execution of all programs assigned to interrupt tasks.


## Remarks

- If an interrupt program is being executed while the instruction is executed, the interrupt program is suspended.
- Use **FP_INTERRUPT_CLEAR_REQUESTS** to clear the interrupt signals suspended by the unit while the interrupt is disabled.
- To disable or enable the interrupt for each unit, use the instruction **FP_INTERRUPT_ACTIVATE**.

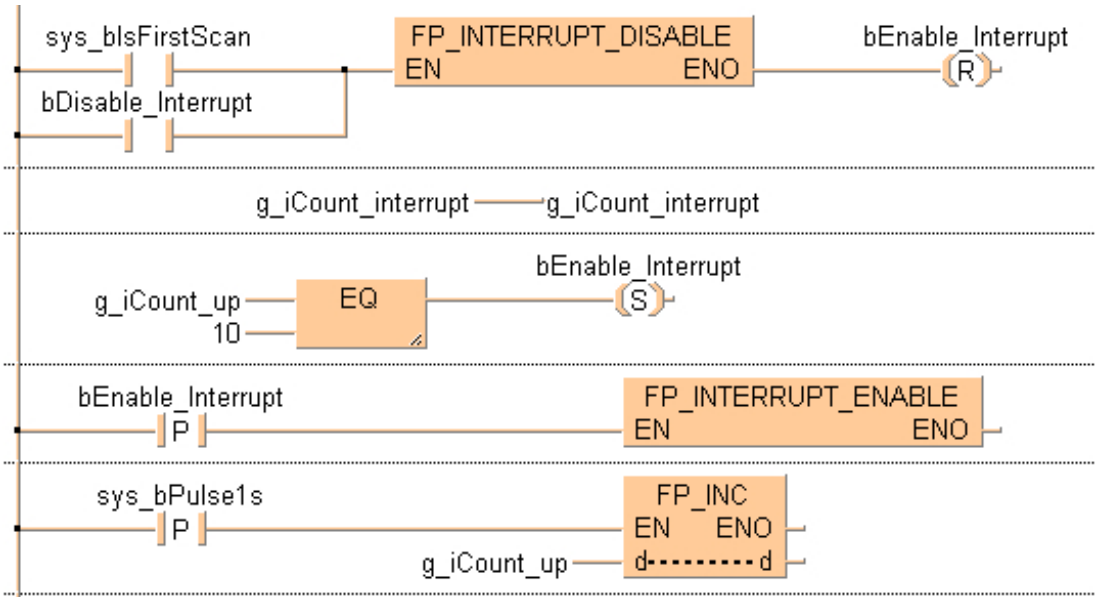
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_iCount_interrupt | INT | 0 |
| 1 | VAR_EXTERNAL | g_bHscUnitEnable | BOOL | FALSE |
| 2 | VAR_EXTERNAL | g_bHscUnitEnableCounting | BOOL | FALSE |
| 3 | VAR_EXTERNAL | g_bHscUnitComparisonMatchFlag | BOOL | FALSE |
| 4 | VAR_EXTERNAL | g_iCount_up | INT | 0 |
| 5 | VAR | bEnable_Interrupt | BOOL | FALSE |
| 6 | VAR | bComparisonMatchMonitor | BOOL | FALSE |
| 7 | VAR | bDisableHscUnit | BOOL | FALSE |
| 8 | VAR | bDisable_Interrupt | BOOL | FALSE |

### LD body


sys_bIsFirstScan
bDisable_Interrupt
FP_INTERRUPT_DISABLE
EN
ENO
bEnable_Interrupt
R
g_iCount_interrupt
g_iCount_interrupt
g_iCount_up
10
EQ
bEnable_Interrupt
S
bEnable_Interrupt
P
FP_INTERRUPT_ENABLE
EN
ENO
sys_bPulse1s
P
FP_INC
EN
ENO
g_iCount_up
d
d

# FP_INTERRUPT_ENABLE

Enable interrupt programs

This FP instruction enables the execution of all programs assigned to interrupt tasks.


## Remarks

- For periodic interrupts, the following applies: After you enable interrupt programs, the first execution of an interrupt program is possible after the time interval that has been defined for the interrupt program has elapsed.
- When the PLC switches from PROG to RUN mode, interrupt programs are disabled. It is necessary to enable interrupt programs by executing **FP_INTERRUPT_ACTIVATE**.

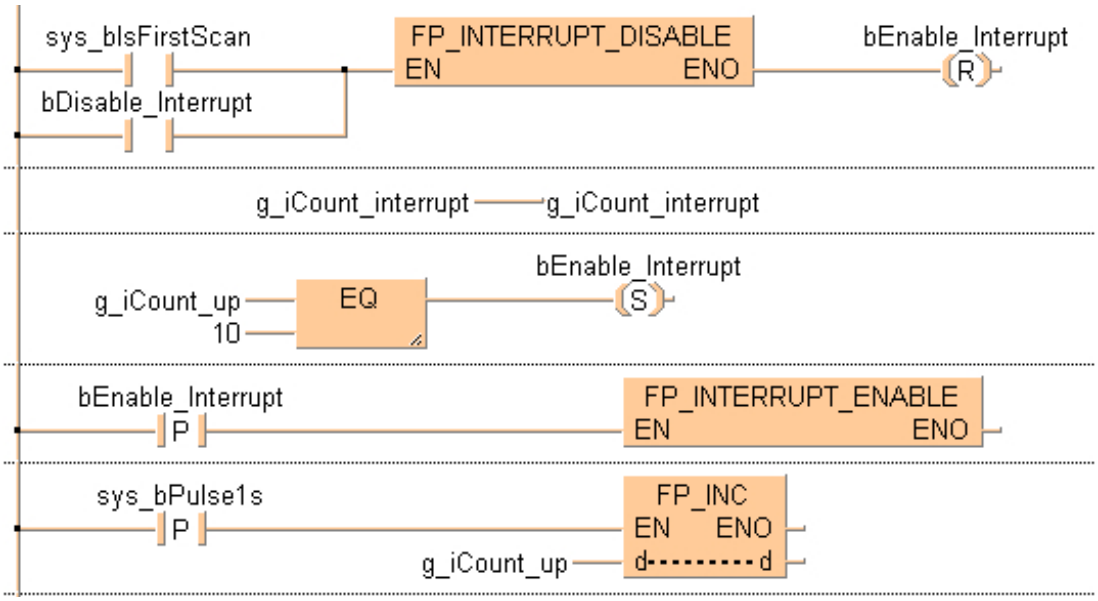
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_iCount_interrupt | INT | 0 |
| 1 | VAR_EXTERNAL | g_bHscUnitEnable | BOOL | FALSE |
| 2 | VAR_EXTERNAL | g_bHscUnitEnableCounting | BOOL | FALSE |
| 3 | VAR_EXTERNAL | g_bHscUnitComparisonMatchFlag | BOOL | FALSE |
| 4 | VAR_EXTERNAL | g_iCount_up | INT | 0 |
| 5 | VAR | bEnable_Interrupt | BOOL | FALSE |
| 6 | VAR | bComparisonMatchMonitor | BOOL | FALSE |
| 7 | VAR | bDisableHscUnit | BOOL | FALSE |
| 8 | VAR | bDisable_Interrupt | BOOL | FALSE |

### LD body


sys_bIsFirstScan
bDisable_Interrupt
FP_INTERRUPT_DISABLE
EN
ENO
bEnable_Interrupt
R
g_iCount_interrupt
g_iCount_interrupt
g_iCount_up
10
EQ
bEnable_Interrupt
S
bEnable_Interrupt
P
FP_INTERRUPT_ENABLE
EN
ENO
sys_bPulse1s
P
FP_INC
EN
ENO
g_iCount_up
d
d

## 25.5 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

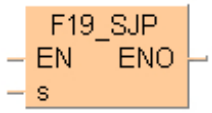
Advantages of IEC instructions

Advantages of FP instructions

# F19_SJP

Indirect jump to label

Jumps to the label **LBLs** with the same number as the data stored in the area specified by **s** if the trigger **EN** is in the ON-state.




## Parameters

Input

**s** (WORD, INT, UINT)

Stores label number (0 to 255)

## Remarks

The range of the number **s** can be between 0 and 255.

## Example

## POU header

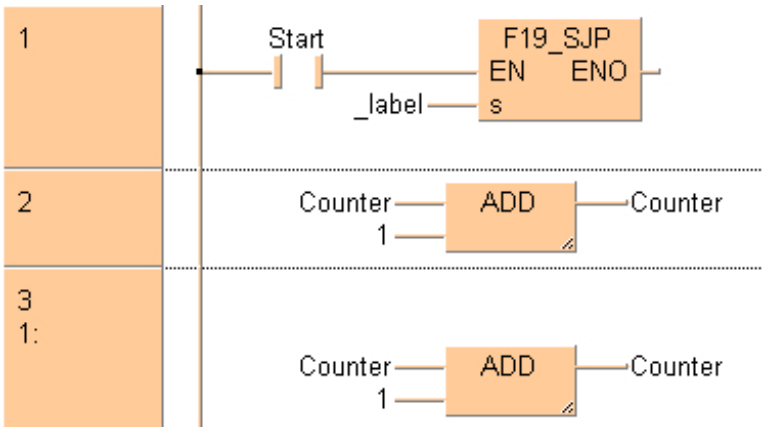
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Start | BOOL | FALSE |
| 1 | VAR | _label | INT | 1 |
| 2 | VAR | Counter | INT | 0 |

## POU body

When the variable **start** is set to TRUE, the function is carried out.

**LD body**


1
Start
F19_SJP
EN
ENO
_label
s
2
Counter
ADD
Counter
1
3
1:
Counter
ADD
Counter
1

# 26 Pulse output instructions

## 26.1 Introduction

Note

The high-speed counter and pulse output instructions can be used with the following FP Series PLCs: FP0, FP-e, FPΣ, FP-X, FP0R.

### Introduction

Control FPWIN Pro offers two concepts for programming with high-speed counter instructions:

- F instructions
- Tool instructions

The tool instructions are universal instructions which are supported by all PLC types of the FP series. They offer new and comfortable features including information functions for evaluating status flags and settings, control functions for configuring high-speed counters and pulse outputs, PLC-independent functions and DUTs, as well as variable channel numbers.
When should you use tool instructions instead of F instructions?

- You want to develop universal function blocks for libraries.
- You must program for different PLC types of the FP series.
- You're tired of setting control code bits and looking up available channel numbers.

However, the F instructions may be easier to use for beginners or users familiar with FPWIN GR.

Most of the information, which is accessible via information and control functions, is stored in special internal flags and special data registers. These flags and registers can also be accessed using PLC-independent system variables.
To take advantage of the features you prefer, the instructions of both libraries can be mixed.

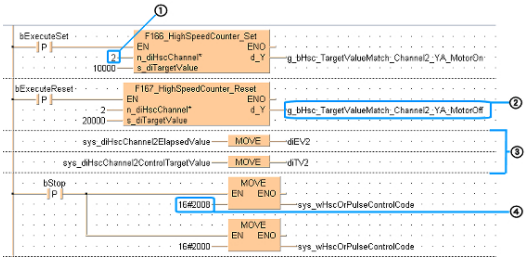
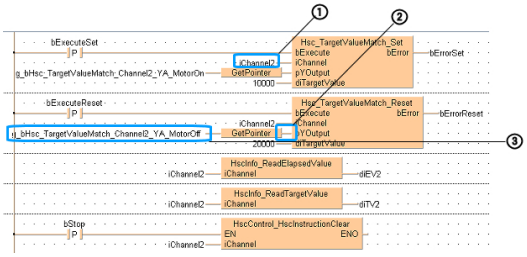
Note

When programming with the tool instructions, be sure to refer to the detailed information provided via the links to the related F instructions.

| Main features | F instructions | Tool instructions |
|---|---|---|
| Pre version 6.4 support | ● | |
| Use of inline functions | ● | |
| Use of FPWIN GR function names | ● | |
| Less code with constant channel numbers | ● | |
| Control codes | ● | |

| Main features | F instructions | Tool instructions |
|---|---|---|
| Control functions | | ● |
| Information functions | | ● |
| Variable channel numbers | | ● |
| Universal functions for all PLCs | | ● |
| For use in universal user function blocks | | ● |
| Common channel configuration DUT for all PLCs and all pulse output instructions | | ● |

## Comparison between programming with F instructions and Tool instructions

| F instructions | | Tool instructions | |
|---|---|---|---|
|  | |  | |
| (1) | Supports only constant channel numbers, in this example channel 2. | (1) | Supports variable channel numbers, in this example channel 2. |
| (2) | Outputs with explicit user addresses in the Y area | (2) | Access to outputs with explicit user addresses via a pointer variable. This pointer variable can also be applied via inputs of user-defined function blocks. |
| (3) | System variables are used to read special data registers for channel 2. | (3) | The name of the output variable **g_bHsc_TargetValueMatch_Channel2_YA_MotorOff** must follow a certain pattern, please refer to Hsc_TargetValueMatch_Set. |
| (4) | PLC-specific control code settings are required, e.g. for clearing a high-speed counter instruction | | |
| Conclusion:<br>• Depends on PLC type<br>• Constant channel number<br>• High maintenance effort, e.g. to change the channel number | | Conclusion:<br>• Independent of PLC type<br>• Variable channel number<br>• Self-explanatory function block names and variables<br>• Can be called by user-defined function blocks with variable channel numbers<br>• Channel number easy to change<br>• Requires more programming steps | |

## Typical applications

Use the high-speed counter instructions to count input pulses from sensors or encoders, and to turn outputs to TRUE or FALSE once a specified target value has been reached.

When used with a motor driver, the pulse output instructions enable typical positioning operations such as trapezoidal control, home return, and JOG operation.

## Specifications

The number of channels for the built-in high-speed counter and for pulse output, the counting range, the input and output numbers, as well as performance specifications differ depending on the PLC type. For details, please refer to the corresponding hardware manual.

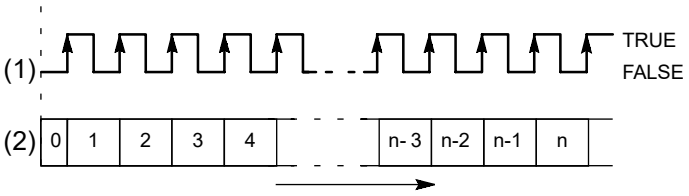
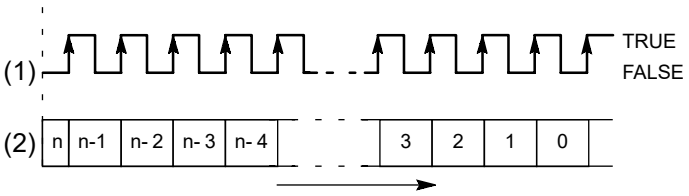
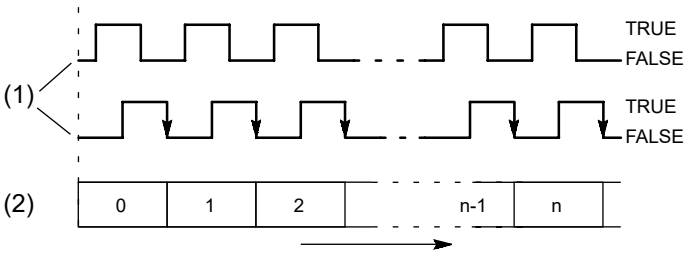
## Required system register settings

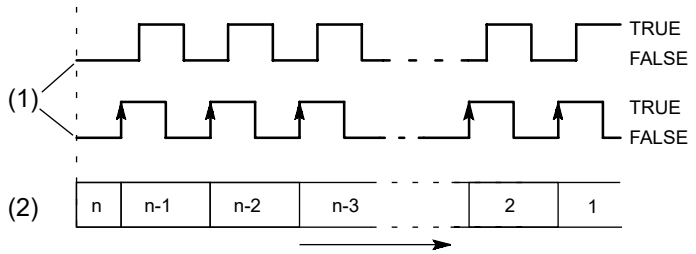
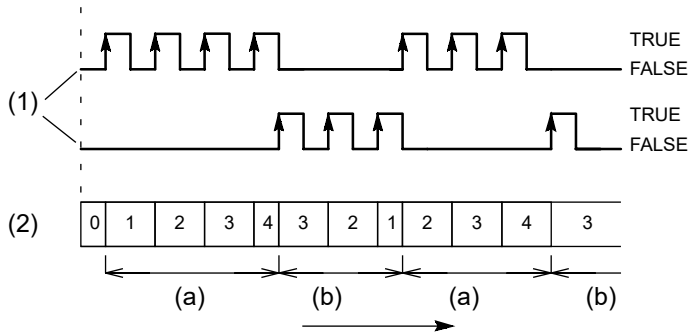
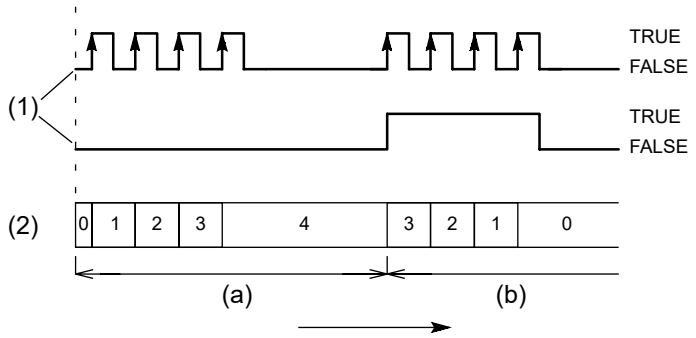
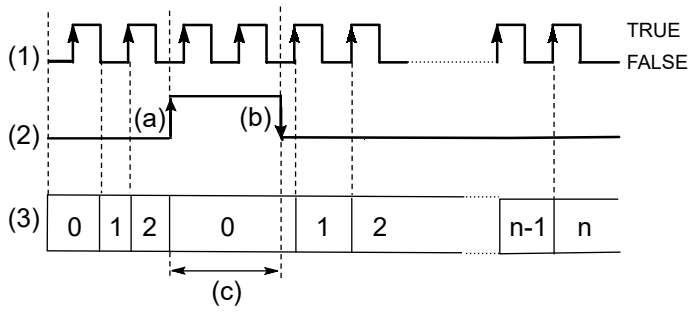
When using a high-speed counter instruction: Select the high-speed counter input for the desired channel in the system registers.

## Counter input modes

To specify the counter input mode, select the high-speed counter inputs in the system registers.

(1) High-speed counter input
(2) Counter value
(3) Reset input

| Input mode | Input signals |
|---|---|
| Incremental |  |
| Decremental |  |
| Two-phase | Incremental counting<br>  |

| Input mode | Input signals |
|---|---|
| | Decremental counting<br>  <br> |
| Incremental/decremental |  <br> |
| | (a) Increasing |
| | (b) Decreasing |
| Incremental/decremental control |  <br> |
| | (a) Increasing |
| | (b) Decreasing |
| Count for reset (incremental) |  <br> |
| | (a) Rising edge: count disabled, elapsed value cleared |
| | (b) Falling edge: count enabled |

| Input mode | Input signals | |
|---|---|---|
| | (c) | Count prohibited |
| | The reset at (3) is executed by the interruption at (a) (rising edge) and (b) (falling edge).The reset input can be enabled/disabled using bit 2 of sys_wHscOrPulseControlCode. | |

### Writing control codes

Control codes are used to perform special counter operations.

- When programming with F instructions:

  Use a MOVE instruction to write or read the control code to or from the special data register reserved for this code (DT90052 or DT9052, depending on the PLC type).The special data register where the high-speed counter and pulse output control code are stored can be accessed with the system variable **sys_wHscOrPulseControlCode**.

- When programming with tool instructions:

  Use universal high-speed counter control instructions and pulse output control instructions which apply to all PLC types to make control code settings. Use the high-speed counter information instructions and pulse output information instructions to monitor control code settings.

### Writing and reading the elapsed value

The elapsed value is stored as a double word in the special data registers.

- When programming with F instructions:
  - Access the special data registers using the system variable **sys_diHscChannelxElapsedValue** (where x=channel number).
  - The channel number is an input parameter of the high-speed counter or pulse output instruction. Most of the other parameters, e.g. speed and target value, can be specified using predefined DUTs. These DUTs can be found in the `FP Library`.
- When programming with tool instructions:
  - Use universal high-speed counter information and control instructions and pulse information and control instructions which apply to all PLC types to read and write the elapsed value.
  - The channel number and control code settings, e.g. CW/CCW, absolute or relative value control, or duty ratio are specified in a channel configuration DUT common to all PLC types. Other parameters, e.g. speed and target value can be applied directly to the instruction.

### Control flags

The high-speed counter and pulse output status is stored in special internal flags. To access the special internal flags, use the PLC-independent system variables.

When a high-speed counter instruction is executed, the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) for the channel used turns to TRUE. No other high-

speed counter instruction using the same channel can be executed as long as the control flag is TRUE.

When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.

- FP-X, FP0R:

  The high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) are assigned to different special internal flags.

- FP-Sigma, FP0, FP-e:

  The high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE. The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

## 26.2 Table operation mode (FP7, FP-XH, FP0H)

### 26.2.1 FP instructions

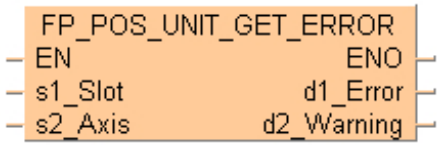
Tip

Advantages of FP instructions

# FP_POS_UNIT_GET_ERROR

Get error or warning in positioning unit

This FP instruction reads the error and warning codes from buffer 1 of the positioning unit in the slot specified by **s1_Slot** and for the axis specified by **s2_Axis** if the trigger **EN** is TRUE. The error code is stored in **d1_Error** and the warning code is stored in **d2_Warning**.




## Parameters

Input

**s1_Slot** (WORD, INT, UINT)

Slot number

**s2_Axis** (WORD, INT, UINT)

Axis number
Values: 1–4, 8 (virtual axis)

Output

**d1_Error** (WORD, INT, UINT)

Error code

**d2_Warning** (WORD, INT, UINT)

Warning code

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the slot and/or axis number is out of range
- if **d1_Error** or **d2_Warning** is out of range

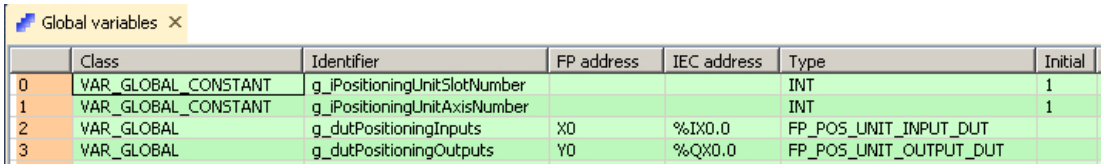
**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the slot and/or axis number is out of range
- if **d1_Error** or **d2_Warning** is out of range

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.


Global variables

| | Class | Identifier | FP address | IEC address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL_CONSTANT | g_iPositioningUnitSlotNumber | | | INT | 1 |
| 1 | VAR_GLOBAL_CONSTANT | g_iPositioningUnitAxisNumber | | | INT | 1 |
| 2 | VAR_GLOBAL | g_dutPositioningInputs | X0 | %IX0.0 | FP_POS_UNIT_INPUT_DUT | |
| 3 | VAR_GLOBAL | g_dutPositioningOutputs | Y0 | %QX0.0 | FP_POS_UNIT_OUTPUT_DUT | |

### POU header

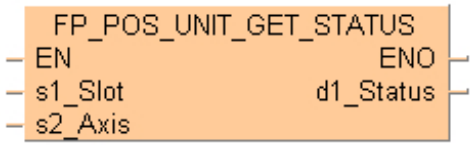
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_dutPositioningInputs | FP_POS_UNIT_INPUT_DUT | |
| 1 | VAR_EXTERNAL_CONSTANT | g_iPositioningUnitSlotNumber | INT | 1 |
| 2 | VAR_EXTERNAL_CONSTANT | g_iPositioningUnitAxisNumber | INT | 1 |
| 3 | VAR | wErrorCodeAxis1 | WORD | 0 |
| 4 | VAR | wWarningCodeAxis1 | WORD | 0 |

# FP_POS_UNIT_GET_STATUS

Get axis status of positioning unit

This FP instruction reads the status data from the positioning unit in the slot specified by **s1_Slot** and for the axis specified by **s2_Axis** if the trigger **EN** is TRUE. The result is stored in **d1_Status**.




## Parameters

Input

**s1_Slot** (WORD, INT, UINT)

Slot number

**s2_Axis** (WORD, INT, UINT)

Axis number
Values: 1–4, 8 (virtual axis)

Output

**d1_Status** (WORD, INT, UINT)

Status information

## Remarks

Types of axis status information

| Bit | Status information | TRUE |
|---|---|---|
| 0 | Tool operation | during tool operation of any axis configured with Configurator PM |
| 1 | Error | if an error occurred on the specified axis |
| 2 | Warning | if a warning was issued for the specified axis |
| 3 | Busy | if the specified axis is operating |
| 4 | Operation done | if operation is completed on the specified axis |
| 5 | Home return done | if home return is completed on the specified axis |

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the slot and/or axis number is out of range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the slot and/or axis number is out of range

## Example

## POU header

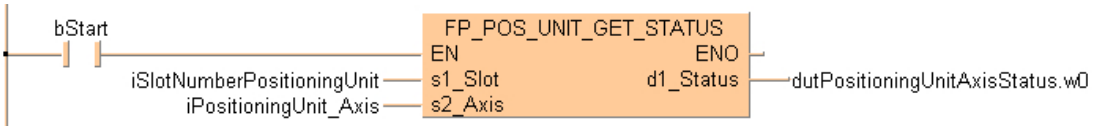
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | |
| 1 | VAR | iSlotNumberPositioningUnit | INT | 0 | slot unit where the positioning unit is assigned ... |
| 2 | VAR | iPositioningUnit_Axis | INT | 0 | Axis of the positioning unit to obtain status for ... |
| 3 | VAR | dutPositioningUnitAxisStatus | BOOL16_OVERLAPPING_DUT | | Status of the positioning unit axis:<br>b0: Tool operation<br>b1: Error annunciation<br>b2: warning annunciation<br>b3: busy<br>b4: operation done<br>b5: home return done |
| 4 | VAR | | | | |

## POU body

When the variable **bStart** is set to TRUE, the function is carried out.

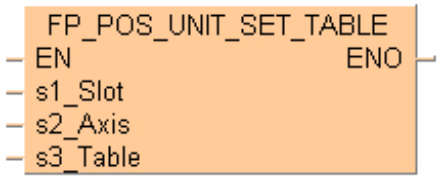
## LD body



# FP_POS_UNIT_SET_TABLE

Set positioning table

This FP instruction starts positioning for the positioning unit in the slot specified by **s1_Slot** and for the axis specified by **s2_Axis** according to the positioning table set at **s3_Table** if the trigger **EN** is TRUE. Configure the positioning table in the Configurator PM7.




## Parameters

Input

**s1_Slot** (WORD, INT, UINT)

Slot number

**s2_Axis** (WORD, INT, UINT)

Axis number
Values: 1–4, 8 (virtual axis)

**s3_Table** (WORD, INT, UINT)

Number of positioning table
Values: 1–600, 10001–10025

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the slot and/or axis number is out of range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the slot and/or axis number is out of range

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global variables

| | Class | Identifier | FP address | IEC address | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL_CONSTANT | g_iPositioningUnitSlotNumber | | | INT | 1 |
| 1 | VAR_GLOBAL_CONSTANT | g_iPositioningUnitAxisNumber | | | INT | 1 |
| 2 | VAR_GLOBAL | g_dutPositioningInputs | X0 | %IX0.0 | FP_POS_UNIT_INPUT_DUT | |
| 3 | VAR_GLOBAL | g_dutPositioningOutputs | Y0 | %QX0.0 | FP_POS_UNIT_OUTPUT_DUT | |

### DUT

- Input

FP_POS_UNIT_INPUT_DUT [DUT]

| | Identifier | Type | Initial |
|---|---|---|---|
| 0 | b00_ReadyPositioning | BOOL | FALSE |
| 1 | b01 | BOOL | FALSE |
| 2 | b02 | BOOL | FALSE |
| 3 | b03 | BOOL | FALSE |
| 4 | b04_ToolOperation | BOOL | FALSE |
| 5 | b05_AxisGroupSettingDone | BOOL | FALSE |
| 6 | b06_AxisParameterSettingDone | BOOL | FALSE |
| 7 | b07_RecalculationDone | BOOL | FALSE |

- Output

FP_POS_UNIT_OUTPUT_DUT [DUT]

| | Identifier | Type | Initial |
|---|---|---|---|
| 0 | b00_SystemStop | BOOL | FALSE |
| 1 | b01 | BOOL | FALSE |
| 2 | b02 | BOOL | FALSE |
| 3 | b03 | BOOL | FALSE |
| 4 | b04 | BOOL | FALSE |
| 5 | b05_AxisGroupSettingChangeRequest | BOOL | FALSE |
| 6 | b06_AxisParameterSettingChangeRequest | BOOL | FALSE |
| 7 | b07_RequestRecalculation | BOOL | FALSE |
| 8 | b08_ServoOnAxis1 | BOOL | FALSE |
| 9 | b09_ServoOnAxis2 | BOOL | FALSE |
| 10 | b0A_ServoOnAxis3 | BOOL | FALSE |
| 11 | b0B_ServoOnAxis4 | BOOL | FALSE |

Copy the DUT to the clipboard and paste the DUT directly into the DUT pool of the navigator to use this DUT in your program.

### POU header

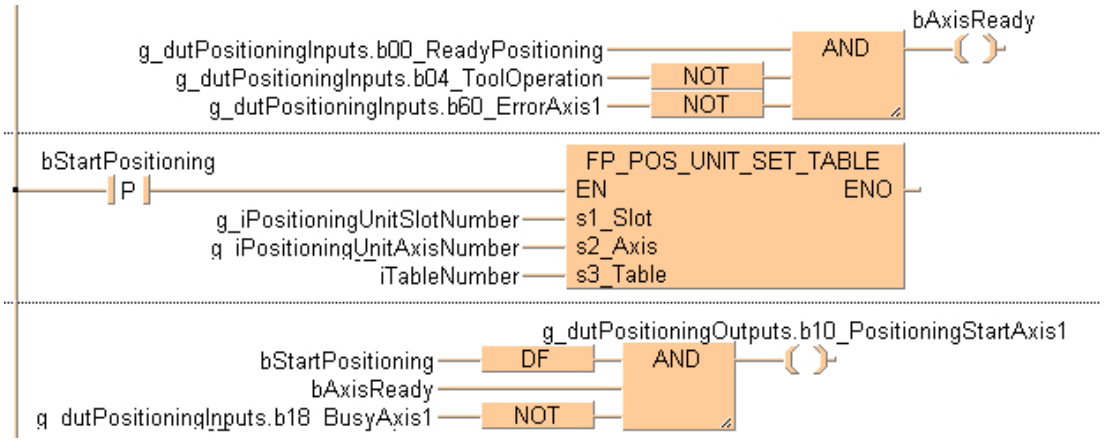
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_dutPositioningInputs | FP_POS_UNIT_INPUT_DUT | |
| 1 | VAR_EXTERNAL | g_dutPositioningOutputs | FP_POS_UNIT_OUTPUT_DUT | |
| 2 | VAR_EXTERNAL_CONSTANT | g_iPositioningUnitSlotNumber | INT | 1 |
| 3 | VAR_EXTERNAL_CONSTANT | g_iPositioningUnitAxisNumber | INT | 1 |
| 4 | VAR | bAxisReady | BOOL | FALSE |
| 5 | VAR | bStartPositioning | BOOL | FALSE |
| 6 | VAR | iTableNumber | INT | 1 |

## POU body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out.

## LD body



### 26.2.2 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

Advantages of IEC instructions

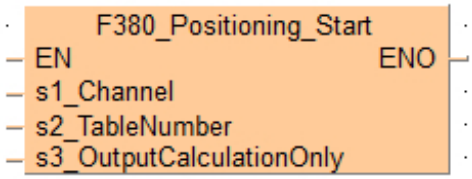
Advantages of FP instructions

# F380_Positioning_Start

Positioning table start

This instruction starts the positioning operation according to the data specified in the positioning memory (positioning table area). The following operations are possible:

- E (end point) control
- P (pass point) control (P → E)
- C (continuation point) control (C → E)
- J (speed point) control (J → E)
- Linear interpolation




## Parameters

Input

**s1_Channel** (WORD, INT, UINT)

Channel number
Configurator PMX: **SYS_PMX_CHANNEL_0**–**SYS_PMX_CHANNEL_3**
Configurator PM7: **SYS_PM7_AXIS_1**–**SYS_PM7_AXIS_8**

**s2_TableNumber** (WORD, INT, UINT)

Table number: 1–20

**s3_OutputCalculationOnly** (WORD, INT, UINT)

Output method:

- 0: pulse output
- 1: calculation only

## Remarks

- If an operand is an out-of-range value, an operation error occurs.
- The stop operation has priority when the conditions of system stop, emergency stop, limit stop and deceleration stop are satisfied.
- An operation error occurs when the system register of a specified channel is other than “Pulse output [Table operation]”.

- A self-diagnostic error (positioning operation error) occurs when the set value or the value of the positioning memory (axis setting area) is abnormal.
- When the channel to be started has been already operating, the positioning control terminates.
- When calculation only is specified for **s3_OutputCalculationOnly**, only the table calculation is executed. When starting the positioning operation for the same channel and the same table from the next scan after executing the calculation, the start-up time of the positioning control is reduced.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if “Pulse output [Table operation]” has not been set in the system register.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if “Pulse output [Table operation]” has not been set in the system register.

## Example

## POU header

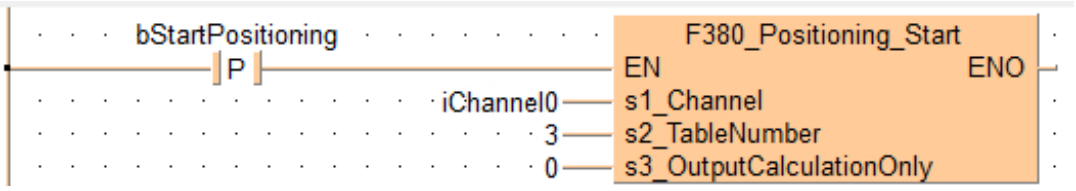
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStartPositioning | BOOL | FALSE |
| 1 | VAR | iChannel0 | INT | 0 |

## POU body

When the variable **bStartPositioning** is set to TRUE, the function is carried out.

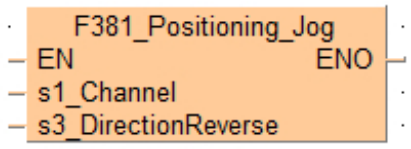
## LD body



# F381_Positioning_Jog

JOG operation start

This instruction starts the JOG operation according to the parameters specified in the positioning memory (axis setting area). While the execution condition is valid, the JOG operation continues.




## Parameters

Input

**s1_Channel** (WORD, INT, UINT)

Channel number
Configurator PMX: **SYS_PMX_CHANNEL_0**–**SYS_PMX_CHANNEL_3**
Configurator PM7: **SYS_PM7_AXIS_1**–**SYS_PM7_AXIS_8**

**s3_DirectionReverse** (WORD, INT, UINT)

Direction:

- 0: forward
- 1: reverse

## Remarks

- If an operand is an out-of-range value, an operation error occurs.
- The stop operation has priority when the conditions of system stop, emergency stop, limit stop and deceleration stop are satisfied.
- An operation error occurs when the system register of a specified channel is other than “Pulse output [Table operation]”.
- A self-diagnostic error (positioning operation error) occurs when the set value or the value of the positioning memory (axis setting area) is abnormal.
- The JOG operation needs to be stopped for switching between the forward rotation and reverse rotation.
- In case of changing a speed, when the target speed after the change is an out-of-range value, the speed change is not executed and the operation continues.
- The target speed can be changed by rewriting the positioning parameter area with a user program. The change is executed after it becomes a constant speed.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if “Pulse output [Table operation]” has not been set in the system register.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if “Pulse output [Table operation]” has not been set in the system register.

## Example

### POU header

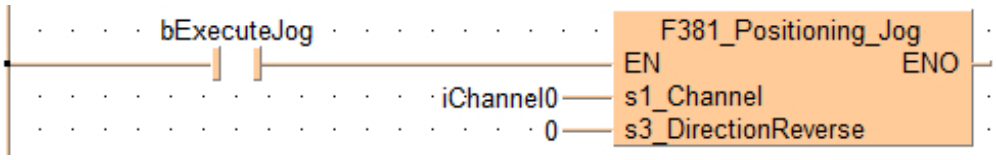
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bExecuteJog | BOOL | FALSE |
| 1 | VAR | iChannel0 | INT | 0 |

### POU body

When the variable **bExecuteJog** is set to TRUE, the function is carried out.

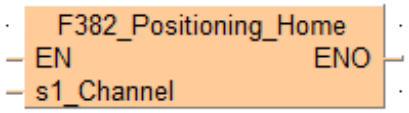
### LD body



# F382_Positioning_Home

Home return

This instruction starts the home return operation according to the parameters specified in the positioning memory (axis setting area).




## Parameters

Input

**s1_Channel** (INT)

Channel number
Configurator PMX: **SYS_PMX_CHANNEL_0**–**SYS_PMX_CHANNEL_3**
Configurator PM7: **SYS_PM7_AXIS_1**–**SYS_PM7_AXIS_8**

## Remarks

- If an operand is an out-of-range value, an operation error occurs.
- The stop operation has priority when the conditions of system stop, emergency stop, limit stop and deceleration stop are satisfied.
- An operation error occurs when the system register of a specified channel is other than “Pulse output [Table operation]”.
- When the home return pattern is “DOG method 1”, “DOG method 3” or “Home return method”, an operation error occurs if the home input has not been set in the system register. When the home return pattern is “DOG method 2” or “Data set method”, the home return starts even if the home input has not been set in the system register.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if “Pulse output [Table operation]” has not been set in the system register.

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if a value specified for a parameter is outside the permissible range.
- if “Pulse output [Table operation]” has not been set in the system register.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bGoHome | BOOL | FALSE |
| 1 | VAR | iChannel0 | INT | 0 |

### POU body

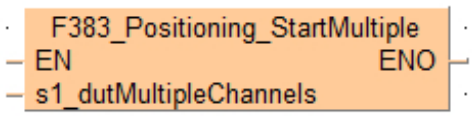
When the variable **bGoHome** changes from FALSE to TRUE, the function is carried out.

### LD body

# F383_Positioning_StartMultiple

Simultaneous start of multiple positioning tables

This instruction starts multiple positioning tables simultaneously. The numbers of the positioning tables are specified by **s1_dutMultipleChannels**. Each positioning table controls a single axis as specified with Configurator PMX. The tables of the E point control, P point control and C point control can be started.




## Parameters

Input

**s1_dutMultipleChannels** (F383_MultipleChannels_DUT)

The starting area of the data register storing the data table numbers to be started simultaneously

## Remarks

- If an operand is an out-of-range value, an operation error occurs.
- The stop operation has priority when the conditions of system stop, emergency stop, limit stop and deceleration stop are satisfied.
- An operation error occurs when the system register of a specified channel is other than “Pulse output [Table operation]”.
- Only when all the specified channels can be started, they are executed simultaneously. When the status of any of the specified channels is "busy", the positioning tables are not started simultaneously and the process is terminated. Use **FP_POS_UNIT_GET_STATUS** to find out the status of each channel.
- Use **F380_Positioning_Start** to start linear interpolation. When the table of the interpolation axis control has been specified with **F383_Positioning_StartMultiple**, a self-diagnostic error (positioning operation error) occurs.
- **s1_dutMultipleChannels** will start the specified positioning table number of each channel.
- You can only specify positioning tables performing single-axis control.
- The valid range for positioning table numbers is 0–20. If 0 is specified as the positioning table number, the channel is excluded from the simultaneous start.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the channel or positioning table number is outside the permissible range.
- if “Pulse output [Table operation]” has not been set in the system register.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the channel or positioning table number is outside the permissible range.
- if “Pulse output [Table operation]” has not been set in the system register.

## Example

### POU header

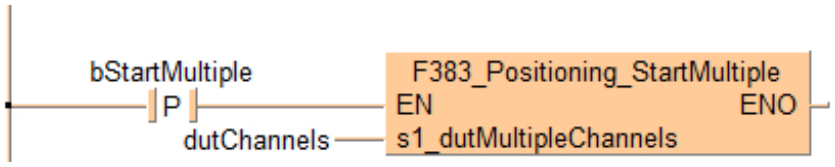
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStartMultiple | BOOL | FALSE |
| 1 | VAR | dutChannels | F383_MultipleChannels_DUT | |

### POU body

When the variable **bStartMultiple** changes from FALSE to TRUE, the function is carried out.

### LD body



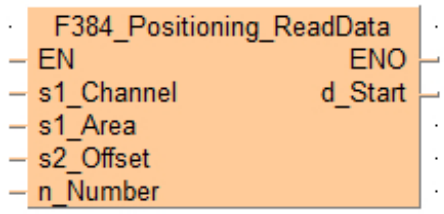
# F384_Positioning_ReadData

Read positioning parameters

This instruction is used to read the following positioning parameters and positioning table data with user programs:

- General information such as channels/axes used, repetition numbers, and error codes
- Axis information such as current execution status and current repetition number
- Axis setting information such as pulse output control codes, home return settings, and speed, acceleration and deceleration settings
- Positioning table data such as control codes and patterns, speed, acceleration and deceleration settings

This instruction reads the number of words specified by **n_Number** of the data stored in the positioning memory starting with **s2_Offset** and stores it in the operation memory area starting with **d_Start**.




## Parameters

Input

**s1_Channel** (WORD, INT, UINT)

Channel number
Configurator PMX: **SYS_PMX_CHANNEL_0**–**SYS_PMX_CHANNEL_3**
Configurator PM7: **SYS_PM7_AXIS_1**–**SYS_PM7_AXIS_8**

**s1_Area** (WORD, INT, UINT)

Positioning memory area:

- 0: **SYS_POSITIONING_AREA_COMMON_DATA**
- 1: **SYS_POSITIONING_AREA_AXIS_INFORMATION**
- 2: **SYS_POSITIONING_AREA_AXIS_SETTING**
- 3: **SYS_POSITIONING_AREA_TABLE_DATA** (FP-XH standard types)
  **SYS_POSITIONING_AREA_CAM_PATTERN** (FP-XH M4T16T, FP-XH M8N16T only)
- 4: **SYS_POSITIONING_AREA_SYNCRONOUS_CONTROL** (FP-XH M4T16T, FP-XH M8N16T only)

- 5: **SYS_POSITIONING_AREA_OPERATION_CHANGE** (FP-XH M4T16T, FP-XH M8N16T only)
- 6: **SYS_POSITIONING_AREA_RTEX_PARAMETER** (FP-XH M4T16T, FP-XH M8N16T only)

**s2_Offset** (WORD, INT, UINT)

Starting offset address of positioning memory areas in the control unit to be read (source address)

**n_Number** (WORD, INT, UINT)

Number of words to be read

Output

**d_Start** (WORD)

Starting address of operation memory in the control unit for storing data read (destination address)

## Example

### POU header

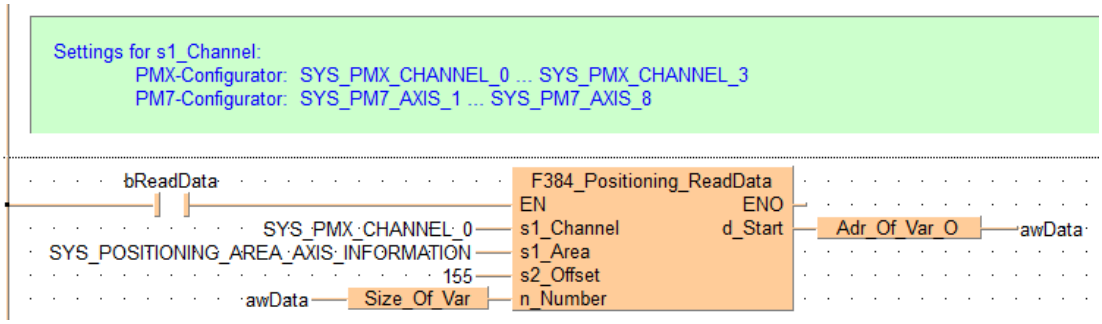
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bReadData | BOOL | FALSE |
| 1 | VAR | awData | ARRAY [0..9] OF WORD | [10(0)] |

### POU body

When the variable **bReadData** is set to TRUE, the function is carried out.

### LD body



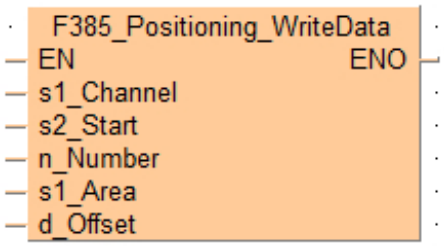
# F385_Positioning_WriteData

Write positioning parameters

This instruction is used to write the following positioning parameters and positioning table data with user programs:

- General information such as channels/axes used, repetition numbers, and error codes
- Axis information such as current execution status and current repetition number
- Axis setting information such as pulse output control codes, home return settings, and speed, acceleration and deceleration settings
- Positioning table data such as control codes and patterns, speed, acceleration and deceleration settings

This instruction writes the number of words (**n_Number**) of the data stored in the operation memory area starting with **s2_Start** and stores it in the positioning memory area starting with **d_Offset**.




## Parameters

Input

**s1_Channel** (WORD, INT, UINT)

Channel number
Configurator PMX: **SYS_PMX_CHANNEL_0**–**SYS_PMX_CHANNEL_3**
Configurator PM7: **SYS_PM7_AXIS_1**–**SYS_PM7_AXIS_8**

**s2_Start** (WORD, INT, UINT)

Starting address of operation memory areas in the control unit to be written (source address)

**n_Number** (WORD, INT, UINT)

Number of words to be written

**s1_Area** (WORD, INT, UINT)

Positioning memory area:

- 0: **SYS_POSITIONING_AREA_COMMON_DATA**

- 1: **SYS_POSITIONING_AREA_AXIS_INFORMATION**
- 2: **SYS_POSITIONING_AREA_AXIS_SETTING**
- 3:
  - **SYS_POSITIONING_AREA_TABLE_DATA** (FP-XH standard types)
  - **SYS_POSITIONING_AREA_CAM_PATTERN** (FP-XH M4T16T, FP-XH M8N16T only)
- 4: **SYS_POSITIONING_AREA_SYNCRONOUS_CONTROL** (FP-XH M4T16T, FP-XH M8N16T only)
- 5: **SYS_POSITIONING_AREA_OPERATION_CHANGE** (FP-XH M4T16T, FP-XH M8N16T only)
- 6: **SYS_POSITIONING_AREA_RTEX_PARAMETER** (FP-XH M4T16T, FP-XH M8N16T only)

**d_Offset** (WORD, INT, UINT)

Starting offset address of the positioning memory in control unit for storing data written (destination address)

## Example

### POU header

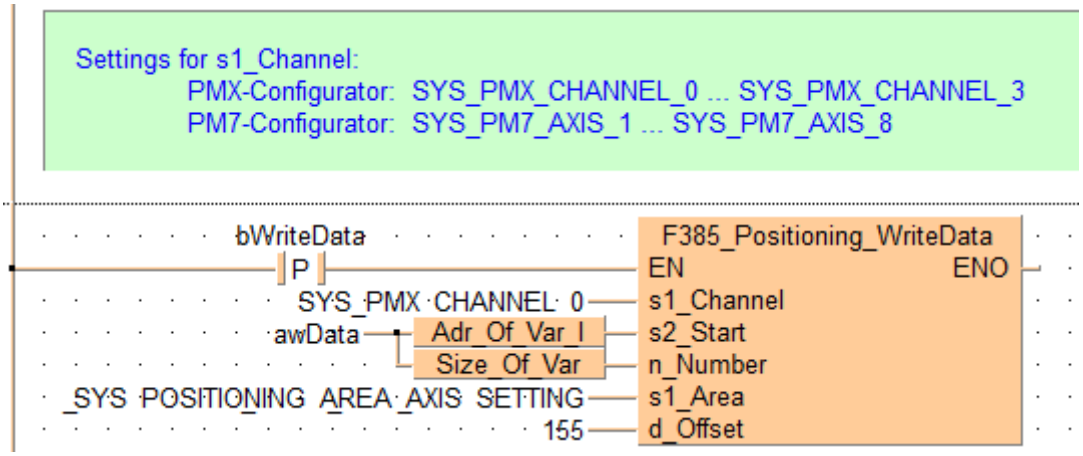
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bWriteData | BOOL | FALSE |
| 1 | VAR | awData | ARRAY [0..9] OF WORD | [10(0)] |

### POU body

When the variable **bWriteData** is set to TRUE, the function is carried out.

### LD body


Settings for s1_Channel:
PMX-Configurator: SYS_PMX_CHANNEL_0 ... SYS_PMX_CHANNEL_3
PM7-Configurator: SYS_PM7_AXIS_1 ... SYS_PM7_AXIS_8
bWriteData
P
SYS_PMX_CHANNEL_0
awData
Adr_Of_Var_I
Size_Of_Var
_SYS_POSITIONING_AREA_AXIS_SETTING
155
F385_Positioning_WriteData
EN
ENO
s1_Channel
s2_Start
n_Number
s1_Area
d_Offset

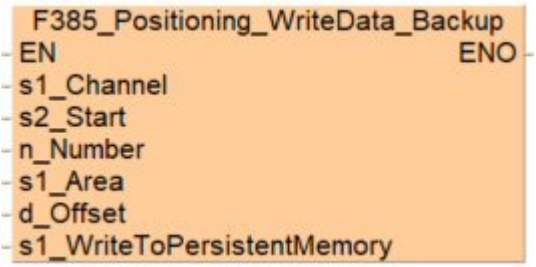
# F385_Positioning_WriteData_Backup

Write positioning parameters with backup

This instruction is used to write the following positioning parameters and positioning table data with user programs:

- General information such as channels/axes used, repetition numbers, and error codes
- Axis information such as current execution status and current repetition number
- Axis setting information such as pulse output control codes, home return settings, and speed, acceleration and deceleration settings
- Positioning table data such as control codes and patterns, speed, acceleration and deceleration settings

This instruction writes the number of words (**n_Number**) of the data stored in the area starting with **s2_Start** and stores it in the positioning memory area starting with **d_Offset**. To keep the written data even after the power has been turned off, the data can additionally be backed up in a persistent memory area.




## Parameters

Input

**s1_Channel** (WORD, INT, UINT)

Channel number
Configurator PMX: **SYS_PMX_CHANNEL_0**–**SYS_PMX_CHANNEL_3**
Configurator PM7: **SYS_PM7_AXIS_1**–**SYS_PM7_AXIS_8**

**s2_Start** (WORD, INT, UINT)

Starting address of operation memory areas in the CPU in which the data to be written are stored (source address)

**n_Number** (WORD, INT, UINT)

Number of words to be written

**s1_Area** (WORD, INT, UINT)

Positioning memory area:

| | |
|---|---|
| 0 | **SYS_POSITIONING_AREA_COMMON_DATA** |
| 1 | **SYS_POSITIONING_AREA_AXIS_INFORMATION** |
| 2 | **SYS_POSITIONING_AREA_AXIS_SETTING** |
| 3 | **SYS_POSITIONING_AREA_TABLE_DATA** |

**d_Offset** (WORD, INT, UINT)

Starting address offset of the positioning memory in the CPU for storing data written (destination address)

**s1_WriteToPersistentMemory** (WORD, INT, UINT)

- TRUE: Data is backed up in the hold area

NOTICE

Possible damage to FROM
When specifying 16#80 to 16#83 (Save in FROM) for the higher 8 bits of input parameter **s1_Channel** (the most significant bit is 1), specified data is written into the CPU's FROM. Writing to FROM can be performed up to 10000 times. We recommend using a rising edge to prevent the writing to FROM from being executed continuously.

- FALSE: Data is not backed up

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the channel number specified at **s1_Channel** is invalid
- if the positioning memory offset specified at **d_Offset** exceeds the limit of the positioning area

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the channel number specified at **s1_Channel** is invalid
- if the positioning memory offset specified at **d_Offset** exceeds the limit of the positioning area

## Example

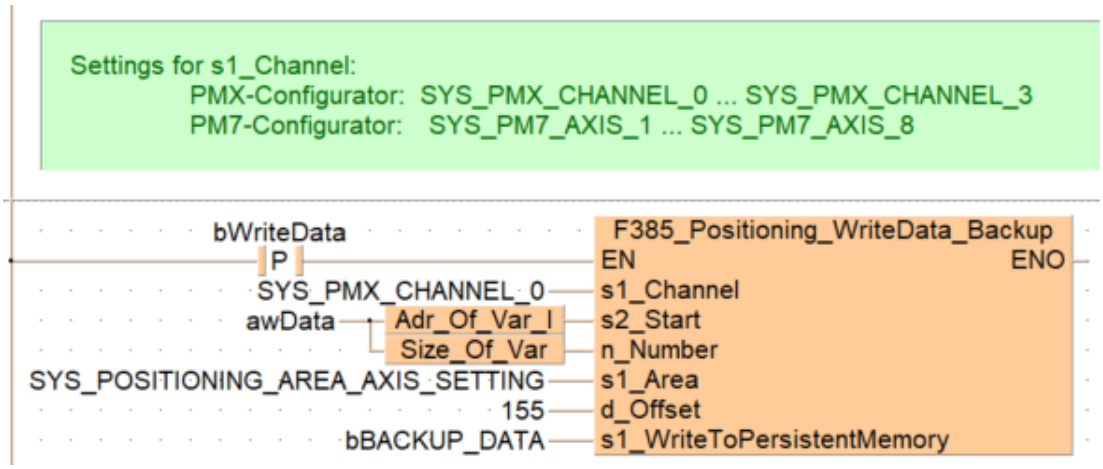
## POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bWriteData | BOOL | FALSE |
| 2 | VAR | awData | ARRAY [0..9] OF WORD | [10(0)] |
| 3 | VAR_CONSTANT | bBACKUP_DATA | BOOL | TRUE |

## LD body

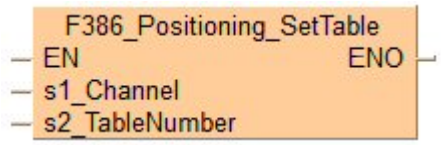
When the variable **bWriteData** changes from FALSE to TRUE, the function is carried out.



# F386_Positioning_SetTable

Set positioning table (FP0H, FP-XH not M4/M8 type)

This instruction starts positioning using the channel specified at **s1_Channel** according to the positioning table set at **s2_TableNumber** if the trigger **EN** is TRUE. Configure the positioning table in the Configurator PM7.




## Parameters

Input

**s1_Channel** (WORD, INT, UINT)

Channel number
Configurator PMX: **SYS_PMX_CHANNEL_0**–**SYS_PMX_CHANNEL_3**
Configurator PM7: **SYS_PM7_AXIS_1**–**SYS_PM7_AXIS_8**

**s2_TableNumber** (WORD, INT, UINT)

number of positioning table
range: 1–600, 10001–10025

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the slot and/or axis number is out of range

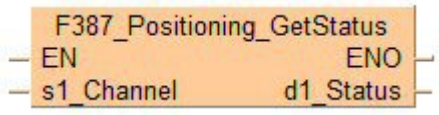
**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the slot and/or axis number is out of range

# F387_Positioning_GetStatus

Get axis status of positioning unit (FP0H, FP-XH not M4/M8 type)

This instruction reads the status data from the positioning unit using the channel specified by **s1_Channel** if the trigger **EN** is TRUE. The result is stored in **d1_Status**.




## Parameters

Input

**s1_Channel** (WORD, INT, UINT)

Channel number
Configurator PMX: **SYS_PMX_CHANNEL_0**–**SYS_PMX_CHANNEL_3**
Configurator PM7: **SYS_PM7_AXIS_1**–**SYS_PM7_AXIS_8**

Output

**d1_Status** (WORD, INT, UINT)

Status information

## Remarks

Types of axis status information

| Bit | Status information | TRUE |
|---|---|---|
| 0 | Tool operation | during tool operation of any axis configured with Configurator PM |
| 1 | Error | if an error occurred on the specified axis |
| 2 | Warning | if a warning was issued for the specified axis |
| 3 | Busy | if the specified axis is operating |
| 4 | Operation done | if operation is completed on the specified axis |
| 5 | Home return done | if home return is completed on the specified axis |

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the slot and/or axis number is out of range

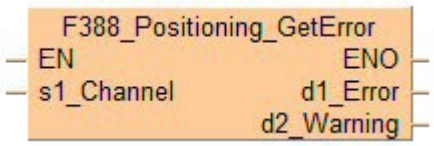
**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the slot and/or axis number is out of range

# F388_Positioning_GetError

Get error or warning in positioning unit (FP0H, FP-XH not M4/M8 type)

This instruction reads the error and warning codes from buffer 1 of the positioning unit using the channel specified by **s1_Channel** if the trigger **EN** is TRUE. The error code is stored in **d1_Error** and the warning code is stored in **d2_Warning**.




## Parameters

Input

**s1_Channel** (WORD, INT, UINT)

Channel number
Configurator PMX: **SYS_PMX_CHANNEL_0**–**SYS_PMX_CHANNEL_3**
Configurator PM7: **SYS_PM7_AXIS_1**–**SYS_PM7_AXIS_8**

Output

**d1_Error** (WORD, INT, UINT)

Error code

**d2_Warning** (WORD, INT, UINT)

Warning code

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the slot and/or axis number is out of range
- if **d1_Error** or **d2_Warning** is out of range

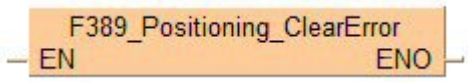
**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the slot and/or axis number is out of range
- if **d1_Error** or **d2_Warning** is out of range

# F389_Positioning_ClearError

Clear error or warning of positioning unit (FP0H, FP-XH not M4/M8 type)

This instruction clears an error or warning in the unit attached if the trigger **EN** is TRUE.



## 26.3 Instruction operation mode (FP0, FP0H, FP-e, FP-Sigma, FP-XH)

### 26.3.1 Introduction to pulse output instructions

The high-speed counter and pulse output instructions can be used with the following FP Series PLCs: FP0, FP-e, FPΣ, FP-X, FP0R.

Note

FP-X, Relay types: The pulse output function is only available if the pulse I/O cassette (AFPX-PLS) has been installed.

#### Typical applications

Use the high-speed counter instructions to count input pulses from sensors or encoders, and to turn outputs to TRUE or FALSE once a specified target value has been reached.

When used with a motor driver, the pulse output instructions enable typical positioning operations such as trapezoidal control, home return, and JOG operation.

#### Specifications

The number of channels for the built-in high-speed counter and for pulse output, the counting range, the input and output numbers, as well as performance specifications differ depending on the PLC type. For details, please refer to the corresponding hardware manual.

#### Required system register settings

When using a high-speed counter instruction: Select the high-speed counter input for the desired channel in the system registers.

#### Position control mode

The position control mode is specified by means of the variables used with the positioning command.

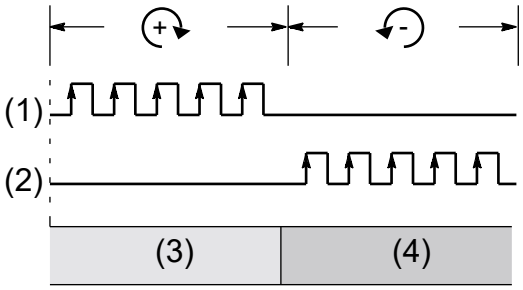
- Relative value control
- Absolute value control

#### Pulse output methods

The pulse output method is specified by means of the variables used with the positioning command.

- CW/CCW

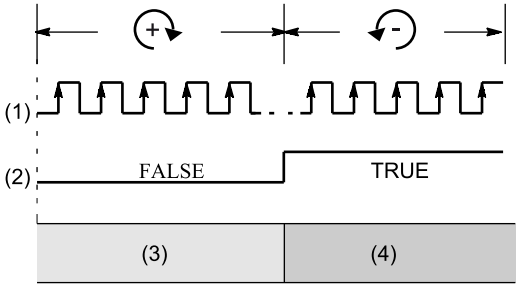
Control is carried out using two pulses: a positive or clockwise rotation pulse (CW) and a negative or counterclockwise rotation pulse (CCW pulse).




(1) CW pulse output
(2) CCW pulse output
(3) Incremental counting
(4) Decremental counting

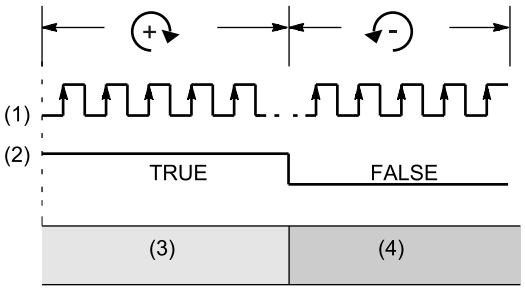
- Pulse/direction
  - Forward FALSE

    In this mode, forward rotation is carried out when the rotation direction signal is FALSE.




  - Forward TRUE

    In this mode, forward rotation is carried out when the rotation direction signal is TRUE.




(1) Pulse output
(2) Direction output
(3) Incremental counting
(4) Decremental counting

### Writing control codes

Control codes are used to perform special counter operations.

- When programming with F instructions:

  Use a MOVE instruction to write or read the control code to or from the special data register reserved for this code (DT90052 or DT9052, depending on the PLC type).The special data register where the high-speed counter and pulse output control code are stored can be accessed with the system variable **sys_wHscOrPulseControlCode**.

- When programming with tool instructions:

  Use universal high-speed counter control instructions and pulse output control instructions which apply to all PLC types to make control code settings. Use the high-speed counter information instructions and pulse output information instructions to monitor control code settings.

### Writing and reading the elapsed value

The elapsed value is stored as a double word in the special data registers.

- When programming with F instructions:
  - Access the special data registers using the system variable **sys_diHscChannelxElapsedValue** (where x=channel number).
  - The channel number is an input parameter of the high-speed counter or pulse output instruction. Most of the other parameters, e.g. speed and target value, can be specified using predefined DUTs. These DUTs can be found in the `FP Library`.
- When programming with tool instructions:
  - Use universal high-speed counter information and control instructions and pulse information and control instructions which apply to all PLC types to read and write the elapsed value.
  - The channel number and control code settings, e.g. CW/CCW, absolute or relative value control, or duty ratio are specified in a channel configuration DUT common to all PLC types. Other parameters, e.g. speed and target value can be applied directly to the instruction.

### Control flags

The high-speed counter and pulse output status is stored in special internal flags. To access the special internal flags, use the PLC-independent system variables.

When a high-speed counter instruction is executed, the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) for the channel used turns to TRUE. No other high-speed counter instruction using the same channel can be executed as long as the control flag is TRUE.

When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.

- FP-X, FP0R:

The high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) are assigned to different special internal flags.

- FP-Sigma, FP0, FP-e:

  The high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE. The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

### Stopping pulse output

- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

## 26.3.2 Writing the pulse output control code

The special data register where the high-speed counter and pulse output control code are stored can be accessed with the system variable **sys_wHscOrPulseControlCode**. (The system variable sys_wHscOrPulseControlCode corresponds to special data register DT90052.)

Operations performed by the pulse output control code:

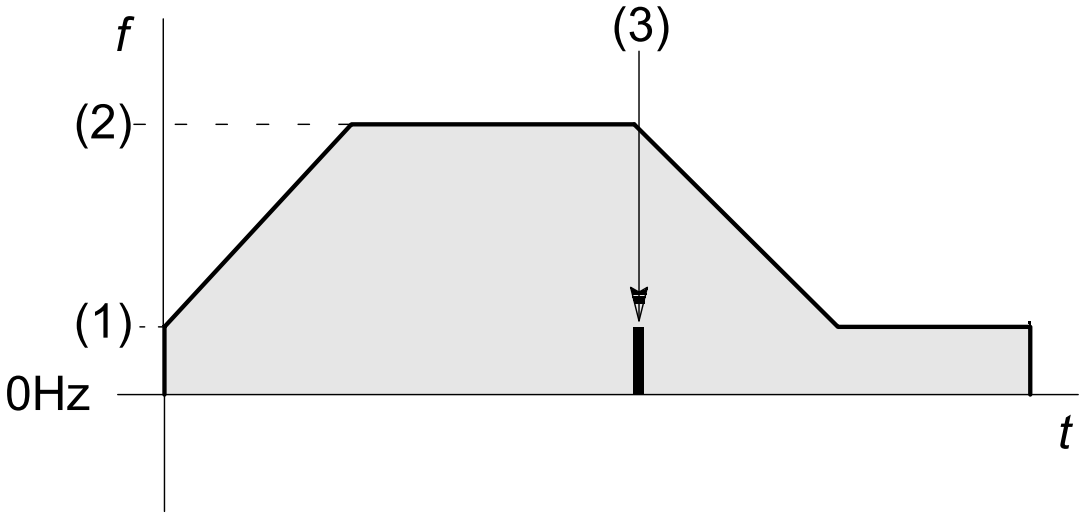
The control code settings for each channel can be monitored using the system variables **sys_wHscChannelxControlCode** or **sys_wPulseChannelxControlCode** (where x=channel number).The settings of this system variable remain unchanged until another setting operation is executed.

Note

- Performing a forced stop may cause the elapsed value at the PLC output side to differ from the elapsed value at the motor input side. Therefore, you must execute a home return after pulse output has stopped.
- Setting the near home input is not possible if counting is prohibited or if a software reset is performed.

## Setting/resetting near home input

To decelerate movement when near the home position, designate a near home input and set bit 4 of the special data register storing the pulse output control code (sys_wHscOrPulseControlCode) to TRUE and back to FALSE again.The near home bit is retained. Set this bit to FALSE right after setting it to TRUE to be able to set the near home input a second time during a home return.

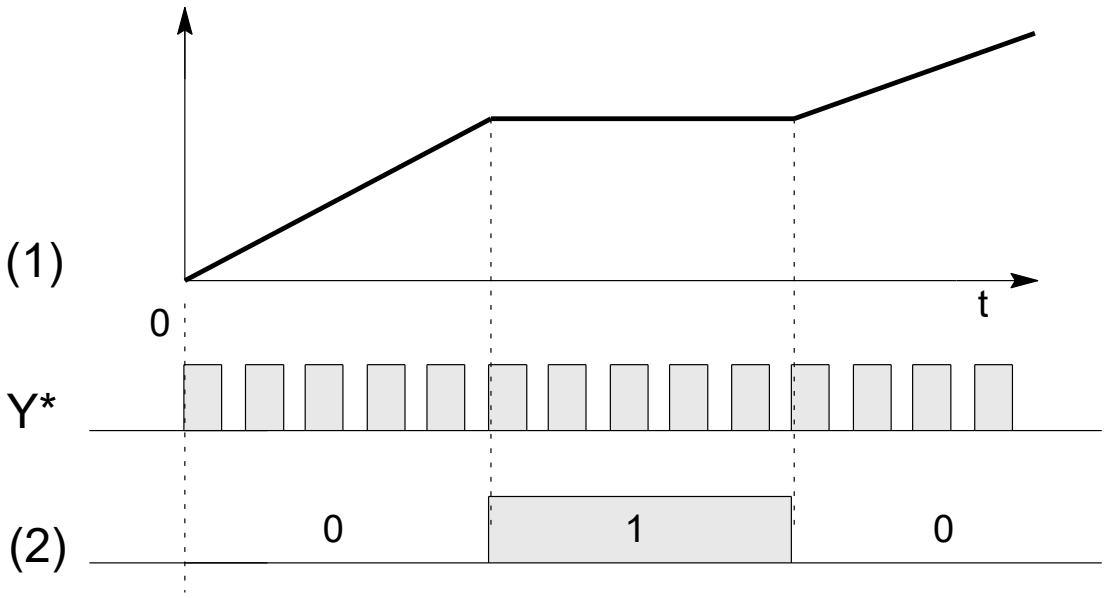



(1) Initial and final speed
(2) Target speed
(3) Near home input: TRUE
(4) Home input: TRUE
(5) Home input is effective at any time.

## Continuing/stopping pulse output (forced stop)

By setting bit 3 of the data register storing the pulse output control code (sys_wHscOrPulseControlCode) to TRUE pulse output is stopped. The possibility of a forced stop should be provided for in every program using pulse output instructions. Reset bit 3 to FALSE to continue pulse output.

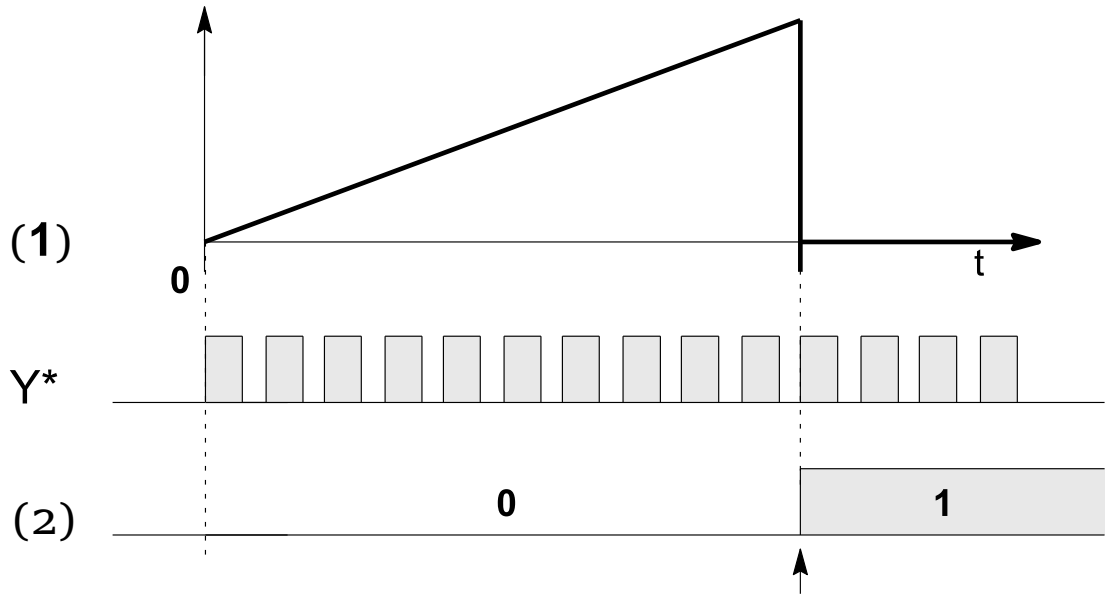
## Enabling/disabling counting operations




| | |
|---|---|
| Y* | Pulse output |
| (1) | Elapsed value |
| (2) | 1 |

When bit 1 of the control code is set to TRUE, counting is prohibited and the elapsed value keeps its current value. Counting is continued when bit 1 is reset to FALSE.

## Resetting the elapsed value (software reset) of the high-speed counter




Y* Pulse output
(1) Elapsed value
(2) Bit 0 of pulse output control code (software reset)

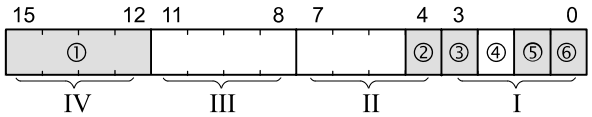
When bit 0 of the control code is set to TRUE, a software reset is performed and the elapsed value is set to 0. The elapsed value keeps the value 0 until bit 0 is reset to FALSE.

## Cancelling high-speed counter and position control instructions (FP0R only)

To cancel execution of a pulse output instruction, set bit 2 of the data register storing the pulse output control code (**sys_wHscOrPulseControlCode**) to TRUE. The pulse output control flag will then change to FALSE. To reenable execution of the instruction, reset bit 2 to FALSE.

## Description for FP-Sigma

Bits 0–15 of the control code are allocated in groups of four. The bit setting in each group is represented by a hex number (e.g. 00020000 0000 1001 = 16#2009).

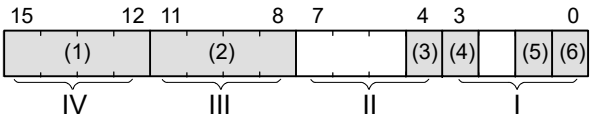


| | | | |
|---|---|---|---|
| Group IV | (1) | Channel number (channel n: 16#n) | |
| Group III | | 0 (fixed) | |
| Group II | (2) | Near home input (bit 4) | |
| | | 0: FALSE | 1: TRUE |
| Group I | (3) | Pulse output (bit 3) | |
| | | 0: continue | 1: stop |
| | (4) | 0 (bit 2, fixed) | |
| | (5) | Count (bit 1) | |
| | | 0: permit | 1: prohibit |
| | (6) | Reset elapsed value to 0 (bit 0) | |
| | | 0: no | 1: yes |

Example: 16#2009

| Group | Value | Description | |
|---|---|---|---|
| IV | 2 | Channel number: 2 | |
| III | 0 | (fixed) | |
| II | 0 | Near home input: FALSE | |
| I | 9 | Hex 9 corresponds to binary 1001 | |
| | | Pulse output: stop (bit 3) | 1 |
| | | (bit 2, fixed) | 0 |
| | | Count: permit (bit 1) | 0 |
| | | Reset elapsed value to 0: yes (bit 0) | 1 |

## Description for FP-X




| | | | |
|---|---|---|---|
| Group IV | (1) | Channel number (channel n: 16#n) | |
| Group III | (2) | 1 (fixed) | |
| Group II | (3) | Near home input (bit 4) (see note) | |
| | | 0: FALSE | 1: TRUE |
| Group I | (4) | Pulse output (bit 3) | |
| | | 0: continue | 1: stop |
| | (5) | Count (bit 1) | |

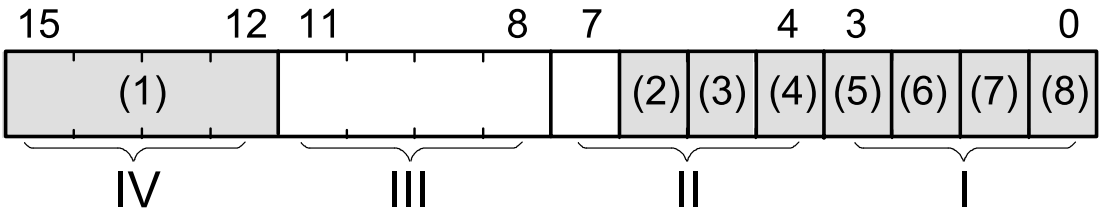
| | | | |
|---|---|---|---|
| | | 0: permit | 1: prohibit |
| | (6) | Reset elapsed value to 0 (bit 0) | |
| | | 0: no | 1: yes |

Example: 16#2109

| Group | Value | Description | |
|---|---|---|---|
| IV | 2 | Channel number: 2 | |
| III | 1 | (fixed) | |
| II | 0 | Near home input: FALSE | |
| I | 9 | Hex 9 corresponds to binary 1001 | |
| | | Pulse output: stop (bit 3) | 1 |
| | | (Bit 2 fixed) | 0 |
| | | Count: permit (bit 1) | 0 |
| | | Reset elapsed value to 0: yes (bit 0) | 1 |

## Description for FP0R

Bits 0–15 of the control code are allocated in groups of four. The bit setting in each group is represented by a hex number (e.g. 00020001 0000 1001 = 16#2109).




| | | | |
|---|---|---|---|
| Group IV | (1) | Channel number (channel n: 16#n) | |
| Group III | | 1 (fixed) | |
| Group II | (2) | Position control start request | |
| | | 0: disabled | 1: enabled |
| | (3) | Decelerated stop request | |
| | | 0: disabled | 1: enabled |
| | (4) | Near home input (bit 4) (see note) | |
| | | 0: FALSE | 1: TRUE |
| Group I | (5) | Pulse output (bit 3) | |
| | | 0: continue | 1: stop |
| | (6) | Cancel pulse output control (bit 2) | |

| | | | |
|---|---|---|---|
| | | 0: continue | 1: stop |
| | (7) | Count (bit 1) | |
| | | 0: permit | 1: prohibit |
| | (8) | Reset elapsed value to 0 (bit 0) | |
| | | 0: no | 1: yes |

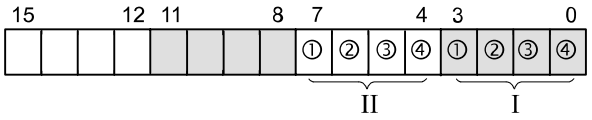
Example: 16#2109

| Group | Value | Description | |
|---|---|---|---|
| IV | 2 | Channel number: 2 | |
| III | 1 | (fixed) | |
| II | 0 | Position control start request: disabled | |
| | | Decelerated stop request: disabled | |
| | | Near home input: FALSE | |
| I | 9 | Hex 9 corresponds to binary 1001 | |
| | | Pulse output: stop (bit 3) | 1 |
| | | Cancel pulse output control (bit 2) | 0 |
| | | Count: permit (bit 1) | 0 |
| | | Reset elapsed value to 0: yes (bit 0) | 1 |

## Description for FP0, FP-e

Bits 0–15 of the control code are allocated in groups of four, each group containing the settings for one channel. The bit setting in each group is represented by a hex number (e.g. 0000 0000 1001 0000 = 16#90).

| Group | II | I |
|---|---|---|
| Channel | 1 | 0 |




(1) Pulse output (bit 3)
0: continue
1: stop

(2) Near home input (bit 2) (see note)
0: FALSE
1: TRUE

(3) Count (bit 1)
0: permit
1: prohibit

(4) Reset elapsed value to 0 (bit 0)
0: no
1: yes

Example: 16#90

| Group | Value | Description | |
|---|---|---|---|
| II | 9 | Channel number: 1<br>Hex 9 corresponds to binary 1001 | |
| | | Pulse output: stop (bit 3) | 1 |
| | | Near home input: FALSE (bit 2) | 0 |
| | | Count: permit (bit 1) | 0 |
| | | Reset elapsed value to 0: yes (bit 0) | 1 |
| I | 0 | – | |

## Example: enable the near home input for channel 2 (FP-SIGMA)

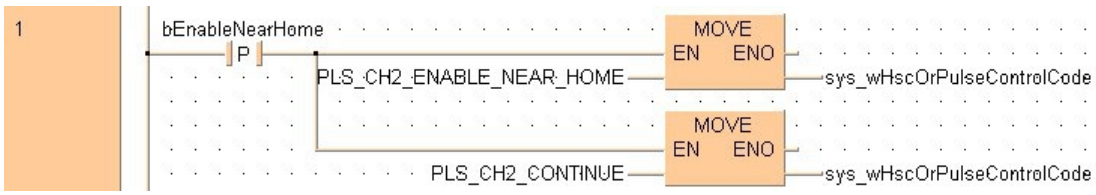
### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bEnableNearHome | BOOL | FALSE | |
| 1 | VAR_CONSTANT | PLS_CH2_ENABLE_NEAR_HOME | WORD | 16#2010 | Enables near home input for channel 2 |
| 2 | VAR_CONSTANT | PLS_CH2_CONTINUE | WORD | 16#2000 | Disables near home input for channel 2 and starts deceleration |
| 3 | VAR | | | | |

### LD body

The near home input is enabled for channel 2 during home return operations.



### Example: performing a forced stop for channel 0 (FP0, FP-e, FP-SIGMA)

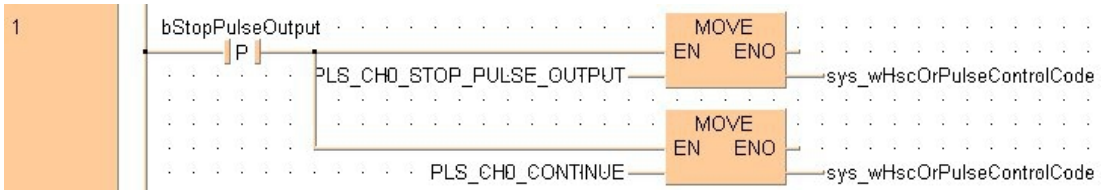
#### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStopPulseOutput | BOOL | FALSE | |
| 1 | VAR_CONSTANT | PLS_CH0_STOP_PULSE_OUTPUT | WORD | 16#0008 | Stop pulse output |
| 2 | VAR_CONSTANT | PLS_CH0_CONTINUE | WORD | 16#0000 | Start preset operation |

#### LD body

A forced stop of the pulse output is performed for channel 0.




Note

Performing a forced stop may cause the elapsed value at the PLC output side to differ from the elapsed value at the motor input side. Therefore, you must execute a home return after pulse output has stopped.

## 26.3.3 Pulse output: writing and reading the elapsed value

The elapsed value is stored as a double word in the special data registers. Access the special data registers using the system variable **sys_diPulseChannelxElapsedValue** (where x=channel number).

For the system variables used for memory areas, please refer to the online help.

System variables for memory areas used

| **Pulse output: elapsed value for channel** | **System variable** | **PLC types** | | |
|---|---|---|---|---|
| | | **FPΣ** | **FP-X, Transistor types FP0** | **FP-X, Relay types FP0R** |
| 0 | **sys_diPulseChannel0ElapsedValue** | • | • | • |
| 1 | **sys_diPulseChannel1ElapsedValue** | | • | • |

| Pulse output: elapsed value for channel | System variable | PLC types | | |
|---|---|---|---|---|
| | | FPΣ | FP-X, Transistor types FP0 | FP-X, Relay types FP0R |
| 2 | **sys_diPulseChannel2ElapsedValue** | ● | ● | |
| 3 | **sys_diPulseChannel3ElapsedValue** | | ● | |

Available channel: ●

## Example: writing the elapsed value into the high-speed counter

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bChangeElapsedValue | BOOL | FALSE | Changes the elapsed value |

### POU body

An initial value of 3000 (elapsed value) is written into channel 0 of the high-speed counter.

### LD body


## Example: reading the elapsed value from pulse output channel 0

### POU header

All input and output variables used for programming this function have been declared in the POU header.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bReadElapsedValue | BOOL | FALSE | Reads the elapsed value |
| 1 | VAR | diElapsedValue | DINT | 0 | Outputs elapsed value |

### POU body

The elapsed value of the high-speed counter is read from channel 0 of the high-speed counter and copied to the variable **diElapsedValue**.

### LD body

### 26.3.4 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

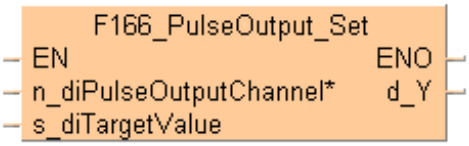
Advantages of IEC instructions

Advantages of FP instructions

# F166_PulseOutput_Set

Target value match ON (pulse output)

If the elapsed value matches the target value of the selected pulse output channel, the specified output immediately turns to TRUE.




## Parameters

Input

**n_diPulseOutputChannel** (DINT)

Pulse output channel:0–3

**s_diTargetValue** (DINT)

specify a 32-bit data value for the target value within the following range: -2147483467 to +2147483648

Output

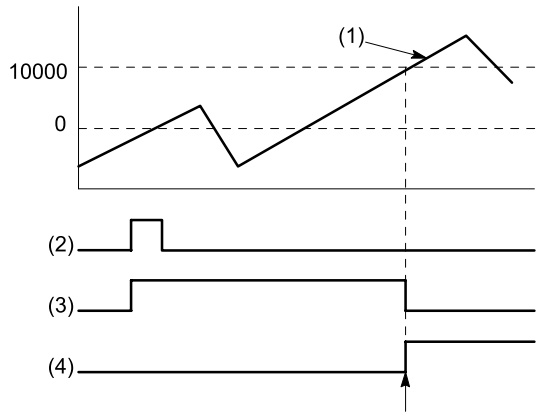
**d_Y** (BOOL)

output which turns to TRUE when the elapsed value matches the target value:Y0–Y1F

## Remarks

- Pulse output characteristics

  Target value: 10000



(1) Elapsed value of pulse output

(2) Execution condition

(3) "Output control active" flag

(4) PLC output
The PLC output turns to TRUE when the elapsed value matches the target value.In addition, the "Output control active" flag turns to FALSE and the instruction is deactivated.If an output is specified that has not been implemented, only the internal memory of the corresponding WY address is set or reset.

- Interrupt operation

  The interrupt program will be executed when the elapsed value matches the target value. Any interrupt that has been entered into the Tasks list is automatically enabled.A special interrupt program number is assigned to each channel number.

- Channels used by interrupt programs:

| | |
|---|---|
| Interrupt8 | Channel0 |
| Interrupt9 | Channel1 |
| Interrupt10 | Channel2 |
| Interrupt11 | Channel3 |

### General programming information

- Set "Pulse output" for the desired channel in the system registers.
- When this instruction is executed, the "Output control active" flag (e.g. **sys_blsPulseChannel0ControlActive**) for the channel used turns to TRUE. No other high-speed counter instruction with output control (F166_PulseOutput_Set or F167_PulseOutput_Reset) using the same channel can be executed as long as this flag is TRUE.
- This instruction is available for all pulse output instructions except F173_PulseOutput_PWM and can be executed before or after execution of a pulse output instruction.
- The duplicate use of an output in other instructions (OUT, SET, RST, KEEP and other F instructions) is not verified by FPWIN Pro and will not be detected.
- To set a PLC output to FALSE that was previously set to TRUE by this instruction, use an RST or MOVE instruction.
- To cancel execution of a pulse output instruction, set bit 2 of the data register storing the pulse output control code (**sys_wHscOrPulseControlCode**) to TRUE. The pulse output control flag will then change to FALSE. To reenable execution of the instruction, reset bit 2 to FALSE.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- We strongly recommend that you incorporate a forced stop option in your positioning program.

- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if channel number or values of the data table are outside the permissible range
- if pulse output has not been set in the system registers

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if channel number or values of the data table are outside the permissible range
- if pulse output has not been set in the system registers

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global Variables

| | Class | Identifier | FP Address | IEC Address | Type | Initial | Comment |
|---|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | out_0 | Y0 | %QX0.0 | BOOL | FALSE | output Y0 of PLC |

### POU header

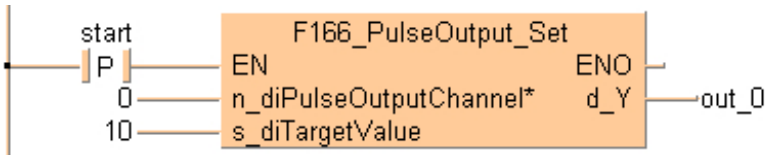
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | out_0 | BOOL | FALSE | output Y0 of PLC |
| 1 | VAR | start | BOOL | FALSE | start condition |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

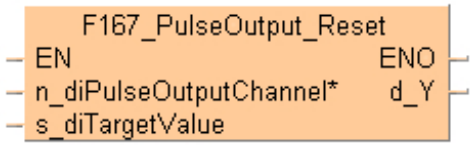
### LD body



# F167_PulseOutput_Reset

Target value match OFF (pulse output)

If the elapsed value matches the target value of the pulse output channel, the specified output immediately turns to FALSE.




## Parameters

Input

**n_diPulseOutputChannel** (DINT)

Pulse output channel: 0–3

**s_diTargetValue** (DINT)

specify a 32-bit data value for the target value within the following range: -2147483467 to +2147483648

Output

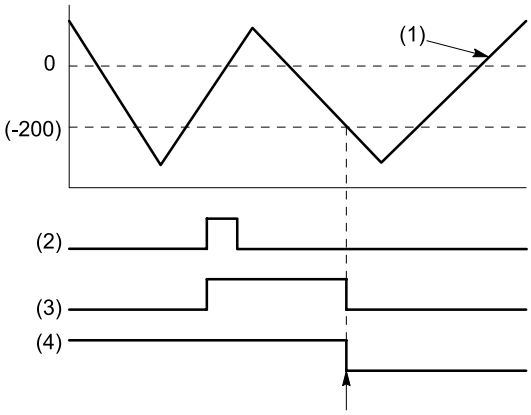
**d_Y** (BOOL)

output which turns to FALSE when the elapsed value matches the target value: Y0–Y1F

## Remarks

Pulse output characteristics

Target value: -200



(1) Elapsed value of pulse output

(2) Execution condition

(3) "Output control active" flag

(4) PLC output
The PLC output turns to FALSE when the elapsed value matches the target value.In addition, the "Output control active" flag turns to FALSE and the instruction is deactivated.If an output is specified that has not been implemented, only the internal memory of the corresponding WY address is set or reset.

Interrupt operation

The interrupt program will be executed when the elapsed value matches the target value. Any interrupt that has been entered into the Tasks list is automatically enabled.A special interrupt program number is assigned to each channel number.

| | |
|---|---|
| Interrupt8 | Channel0 |
| Interrupt9 | Channel1 |
| Interrupt10 | Channel2 |
| Interrupt11 | Channel3 |

## General programming information

- Set "Pulse output" for the desired channel in the system registers.
- When this instruction is executed, the "Output control active" flag (e.g. **sys_bIsPulseChannel0ControlActive**) for the channel used turns to TRUE. No other high-speed counter instruction with output control (F166_PulseOutput_Set or F167_PulseOutput_Reset) using the same channel can be executed as long as this flag is TRUE.
- This instruction is available for all pulse output instructions except F173_PulseOutput_PWM and can be executed before or after execution of a pulse output instruction.
- The duplicate use of an output in other instructions (OUT, SET, RST, KEEP and other F instructions) is not verified by FPWIN Pro and will not be detected.
- To cancel execution of a pulse output instruction, set bit 2 of the data register storing the pulse output control code (**sys_wHscOrPulseControlCode**) to TRUE. The pulse output control flag will then change to FALSE. To reenable execution of the instruction, reset bit 2 to FALSE.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- We strongly recommend that you incorporate aforced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if channel number or values of the data table are outside the permissible range
- if pulse output has not been set in the system registers

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if channel number or values of the data table are outside the permissible range
- if pulse output has not been set in the system registers

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global Variables

| | Class | Identifier | FP Address | IEC Address | Type | Initial | Comment |
|---|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | out_0 | Y0 | %QX0.0 | BOOL | FALSE | output Y0 of PLC |

### POU header

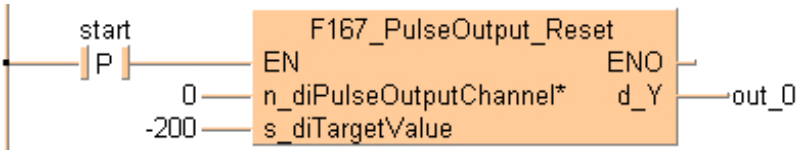
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | out_0 | BOOL | FALSE | output Y0 of PLC |
| 1 | VAR | start | BOOL | FALSE | start condition |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

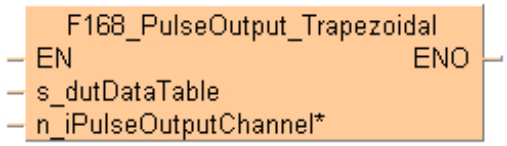
### LD body



# F168_PulseOutput_Trapezoidal

Trapezoidal control

This instruction automatically performs trapezoidal control according to the parameters in the specified DUT.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**s_dutDataTable** (F168_PulseOutput_Trapezoidal_DUT)

Starting address of area containing the data table

**n_iPulseOutputChannel** (decimal constant)

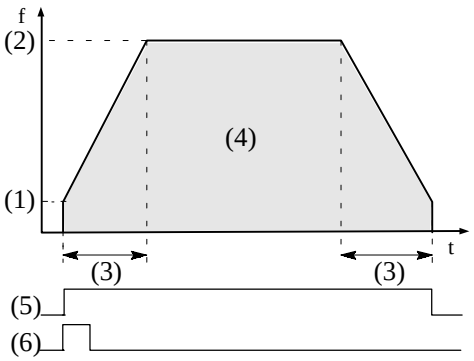
Pulse output: 0 or 1

## Remarks

Use the following predefined DUT: **F168_PulseOutput_Trapezoidal_DUT**

- Control code
- Initial and final speed
- Target speed
- Acceleration/deceleration time
- Target value
- Pulse stop (fixed)

## Pulse output characteristics




(1) Initial and final speed
(2) Target speed
(3) Acceleration/deceleration time
(4) Target value
(5) Pulse output control flag
(6) Execution condition

The pulse output frequency changes according to the specified acceleration/deceleration time. The difference between target and initial speed determines the slope of the ramps.

## General programming information

- Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- The high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- Pulse output stops when the upper limit of the internal elapsed value is exceeded if rotation is in one direction only. As a countermeasure, reset the elapsed value to 0 before executing this instruction. Pulse output does not stop when the FP0R is used in FP0 compatibility mode because the data range for the elapsed value is a signed 32-bit value.
- We strongly recommend that you incorporate a forced stop option in your positioning program.

- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

### Running the FP0R in FP0 compatibility mode

To run the FP0R in FP0 compatibility mode, you can download an FP0 program to the FP0R. Please note the following restrictions:

- The FP0R supports signed 32-bit data for elapsed value and target value; the FP0 supports signed 24-bit data. In FP0 compatibility mode, counting and pulse output continue even if data exceeds the FP0 range.
- The duty ratio is always 25% regardless of the settings in the instructions. With the pulse output method "pulse/direction", pulses are output approx. 300μs after the direction signal has been output; the motor driver characteristics are simultaneously taken into consideration.
- The FP0R does not support the "no counting" setting. Instead, incremental counting is performed with the FP0 pulse output instructions set to "no counting".
- The maximum pulse output frequency is 10000Hz.
- Make sure the pulse output instruction does not use an output that is also being used as a normal output.
- An FP0 program can only run in FP0 compatibility mode, if the PLC types (C10, C14, C16, C32, and T32) match exactly. FP0 compatibility mode is not available for the F32 type FP0R.

### Outputs and system variables for FP0, FP-e

Channel and pulse output numbers

| Channel no. | Pulse output | Pulse output method |
|---|---|---|
| **0** | Y0 | Pulse |
| | Y2 | Direction |
| **1** | Y1 | Pulse |
| | Y3 | Direction |

System variables for memory areas used. Values in parentheses are valid for FP0 T32.

| **Description** | | **System variable** |
|---|---|---|
| Pulse output: control flag for channel | 0 | **sys_bIsPulseChannel0Active** |
| | 1 | **sys_bIsPulseChannel1Active** |
| Pulse output: elapsed value for channel | 0 | **sys_diPulseChannel0ElapsedValue** |
| | 1 | **sys_diPulseChannel1ElapsedValue** |
| Pulse output: target value for channel | 0 | **sys_diPulseChannel0TargetValue** |

| Description | | System variable |
|---|---|---|
| | 1 | **sys_diPulseChannel1TargetValue** |
| High-speed counter or pulse output controlcode | | **sys_wHscOrPulseControlCode** |

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if channel number or values of the data table are outside the permissible range
- if initial speed < 40
- if initial speed > maximum speed

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if channel number or values of the data table are outside the permissible range
- if initial speed < 40
- if initial speed > maximum speed

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global Variables

| | Class | Identifier | FP Address | IEC Address | Type |
|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | X0_bMotorSwitch | X0 | %IX0.0 | BOOL |

### DUT

The DUT F168_PulseOutput_Trapezoidal_DUT is predefined in the FP Library.

### POU header

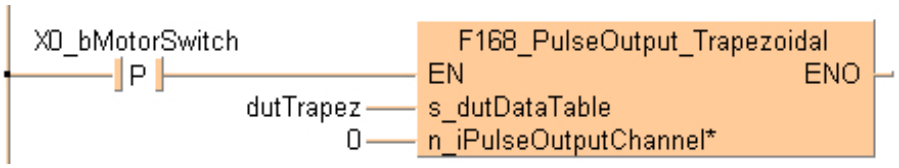
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | dutTrapez | F168_PulseOutput_Trapezoidal_DUT | wControlCode := 16#102,<br>iInitialAndFinalSpeed := 1000,<br>iTargetSpeed := 7000,<br>iAccelerationAndDecelerationTime := 300,<br>diTargetValue := 10000 | |
| 1 | VAR_EXT... | X0_bMotorSwitch | BOOL | | |
| 2 | VAR | | | | |

### POU body

When **X0_bMotorSwitch** turns to TRUE the function is executed.

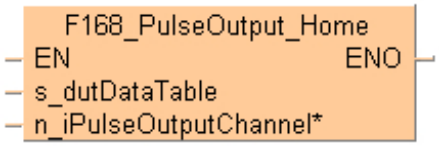
### LD body


X0_bMotorSwitch
P
F168_PulseOutput_Trapezoidal
EN
ENO
dutTrapez
s_dutDataTable
0
n_iPulseOutputChannel*

# F168_PulseOutput_Home

Home return

This instruction performs a home return according to the parameters in the specified DUT.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**s_dutDataTable** (F168_PulseOutput_Home_DUT)

Starting address of area containing the data table

**n_iPulseOutputChannel** (decimal constant)

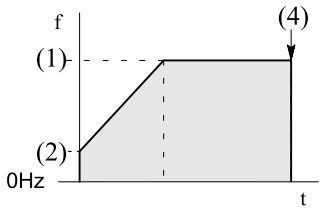
Pulse output: 0 or 1

## Remarks

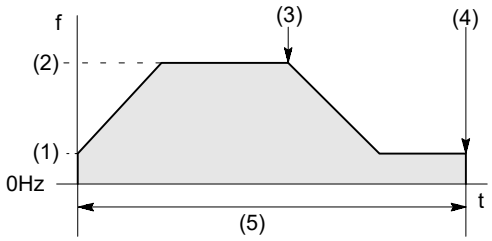
After a drive system has been switched on, there is a difference between the internal position value (elapsed value) and the mechanical position of the axis; this difference cannot be predetermined. The internal value must be synchronized with the actual position value of the axis. This is done by means of a home return, during which a position value is registered at a known reference point (home). During execution of a home return instruction, pulses are continuously output until the home input is enabled. The I/O allocation is determined by the channel used. To decelerate movement when near the home position, designate a near home input and set bit 4 of the special data register storing the pulse output control code (sys_wHscOrPulseControlCode) to TRUE and back to FALSE again. The value in the elapsed value area during a home return differs from the current value. When the return is completed, the elapsed value changes to 0.

Select one of two different operation modes:

- Type 1: The home input is effective regardless of whether or not there is a near home input, whether deceleration is taking place, or whether deceleration has been completed.
  - Without near home input:

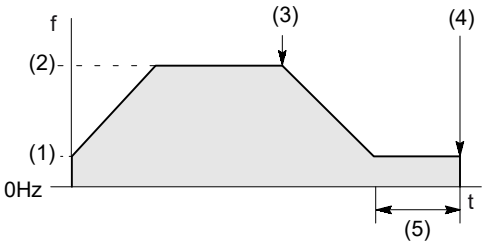



– With near home input:




(1) Initial and final speed
(2) Target speed
(3) Near home input: TRUE
(4) Home input: TRUE
(5) Home input is effective at any time.

- Type 2: The home input is effective only after deceleration (started by near home input) has been completed.




(1) Initial and final speed
(2) Target speed
(3) Near home input: TRUE
(4) Home input: TRUE
(5) Home input is effective only after deceleration

### Use the following predefined DUT:

The following parameters can be specified in the DUT: **F168_PulseOutput_Home_DUT**

- Control code
- Initial and final speed
- Target speed
- Acceleration/deceleration time
- Pulse stop (fixed)

### Pulse output characteristics

- The pulse output frequency changes according to the specified acceleration/deceleration time.
- The difference between target and initial speed determines the slope of the ramps.

### General programming information

- Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.
- The high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- Even when home input has occurred, executing this instruction causes pulse output to begin.
- If the near home input is enabled while acceleration is in progress, deceleration will start.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

### Running the FP0R in FP0 compatibility mode

To run the FP0R in FP0 compatibility mode, you can download an FP0 program to the FP0R. Please note the following restrictions:

- The FP0R supports signed 32-bit data for elapsed value and target value; the FP0 supports signed 24-bit data. In FP0 compatibility mode, counting and pulse output continue even if data exceeds the FP0 range.
- The duty ratio is always 25% regardless of the settings in the instructions. With the pulse output method "pulse/direction", pulses are output approx. 300μs after the direction signal has been output; the motor driver characteristics are simultaneously taken into consideration.
- The FP0R does not support the "no counting" setting. Instead, incremental counting is performed with the FP0 pulse output instructions set to "no counting".
- The maximum pulse output frequency is 10000Hz.

- Make sure the pulse output instruction does not use an output that is also being used as a normal output.
- An FP0 program can only run in FP0 compatibility mode, if the PLC types (C10, C14, C16, C32, and T32) match exactly. FP0 compatibility mode is not available for the F32 type FP0R.

### Outputs and system variables for FP0, FP-e

Channel and pulse output numbers

| Channel no. | Pulse output | Pulse output method |
|---|---|---|
| **0** | Y0 | Pulse |
| | Y2 | Direction |
| **1** | Y1 | Pulse |
| | Y3 | Direction |

System variables for memory areas used. Values in parentheses are valid for FP0 T32.

| **Description** | | **System variable** |
|---|---|---|
| Pulse output: control flag for channel | 0 | **sys_blsPulseChannel0Active** |
| | 1 | **sys_blsPulseChannel1Active** |
| Pulse output: elapsed value for channel | 0 | **sys_diPulseChannel0ElapsedValue** |
| | 1 | **sys_diPulseChannel1ElapsedValue** |
| Pulse output: target value for channel | 0 | **sys_diPulseChannel0TargetValue** |
| | 1 | **sys_diPulseChannel1TargetValue** |
| High-speed counter or pulse output controlcode | | **sys_wHscOrPulseControlCode** |

### Home inputs for FP0, FP-e

| Channel no. | Home input |
|---|---|
| 0 | X0 |
| 1 | X1 |

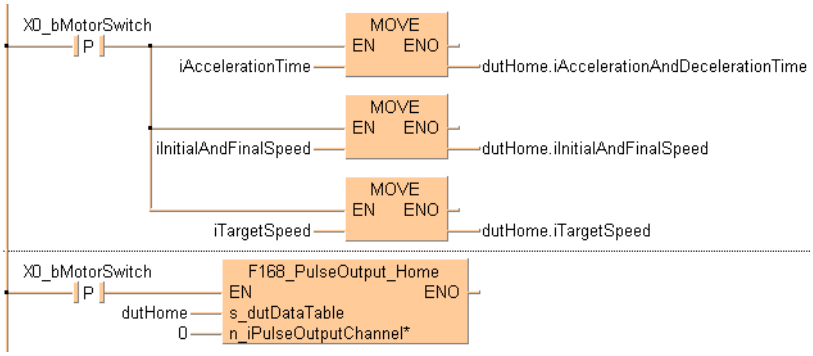
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | X0_bMotorSwitch | BOOL | FALSE | at X0 |
| 1 | VAR | dutHome | F168_PulseOutput_Home_DUT | wControlCode := 16#102, | wControlCode := 16#102,<br>iInitialAndFinalSpeed := 0,<br>iTargetSpeed := 0,<br>iAccelerationAndDecelerationTime := 0 |
| 2 | VAR | iInitialAndFinalSpeed | INT | 3000 | |
| 3 | VAR | iTargetSpeed | INT | 7000 | |
| 4 | VAR | iAccelerationTime | INT | 300 | |

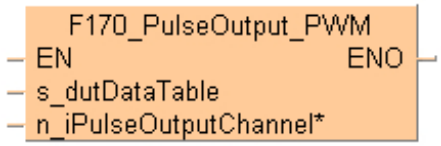
## LD body



# F170_PulseOutput_PWM

PWM output

This instruction delivers a pulse width modulated output signal according to the specified DUT. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**s_dutDataTable** (F170_PulseOutput_PWM_DUT)

Starting address of area containing the data table

**n_iPulseOutputChannel** (INT)

Pulse output: 0 or 1

## Remarks

- Use the following predefined DUT: **F170_PulseOutput_PWM_DUT**
- The following parameters can be specified in the DUT:
  - Approximate frequency
  - Duty ratio (for pulse duration and period)

    The ratio between the pulse duration and the period of a rectangular waveform. For a pulse train in which the pulse duration is 1μs and the pulse period is 4μs, the duty ratio is 0.25 or 25%.

## General programming information

- As soon as you begin editing a program online (i.e., in RUN mode) using this instruction, pulse output will stop.
- Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- The high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter

instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.

- At a point close to the minimum or maximum duty ratio, the output is delayed, which may cause the duty ratio to differ from the specified value.
- The duty ratio can be changed for each scan. The change becomes effective with the next pulse output. The frequency setting is only effective at the start of execution of an instruction.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

### Using the FP0 compatibility mode of the FP0R

To run the FP0R in FP0 compatibility mode, you can download an FP0 program to the FP0R.

### Outputs and system variables for FP0, FP-e

Channel and pulse output numbers

| Channel no. | Pulse output | Pulse output method |
|---|---|---|
| **0** | Y0 | Pulse |
| | Y2 | Direction |
| **1** | Y1 | Pulse |
| | Y3 | Direction |

System variables for memory areas used. Values in parentheses are valid for FP0 T32.

| **Description** | | **System variable** |
|---|---|---|
| Pulse output: control flag for channel | 0 | **sys_blsPulseChannel0Active** |
| | 1 | **sys_blsPulseChannel1Active** |
| Pulse output: elapsed value for channel | 0 | **sys_diPulseChannel0ElapsedValue** |
| | 1 | **sys_diPulseChannel1ElapsedValue** |
| Pulse output: target value for channel | 0 | **sys_diPulseChannel0TargetValue** |
| | 1 | **sys_diPulseChannel1TargetValue** |
| High-speed counter or pulse output controlcode | | **sys_wHscOrPulseControlCode** |

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global Variables

| | Class | Identifier | FP A... | IEC Addr... | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | X6_bEnablePWM | X6 | %IX0.6 | BOOL | FALSE |

### DUT

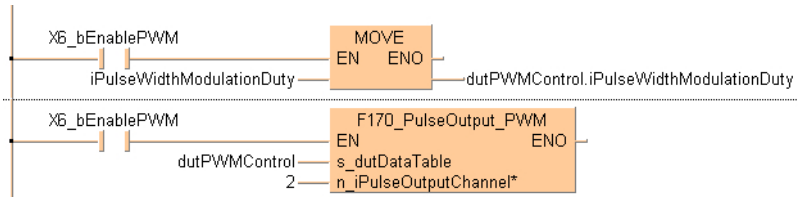
The DUT **F170_PulseOutput_PWM_DUT** is predefined in the `FP Library`.

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXT... | X6_bEnablePWM | BOOL | FALSE | |
| 1 | VAR | dutPWMControl | F170_PulseOutput_PWM_DUT | | iFrequencyValue := 1: f=2.0 Hz, T=502.5 ms; |
| 2 | VAR | iPulseWidthModulationDuty | INT | 500 | 500 = 50% duty |

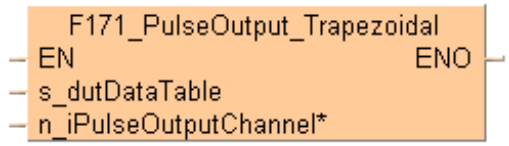
### LD body



# F171_PulseOutput_Trapezoidal

Trapezoidal Control

This instruction automatically performs trapezoidal control according to the parameters in the specified DUT.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




Input

**s_dutDataTable** (DUT)

Starting address of area containing the data table

- FP-Σ, FP-X:
  **F171_PulseOutput_Trapezoidal_DUT**
- FP0R:
  **F171_PulseOutput_Trapezoidal_Type0_DUT**
  **F171_PulseOutput_Trapezoidal_Type1_DUT**

**n_iPulseOutputChannel** (decimal constant)

Pulse output channel:
FP-XH C30 T/P: 0–3
FP-XH C60 T/P: 0–5
FP-Σ: 0, 2
FP-X R: 0, 1
FP-XC14T: 0, 1, 2
FP-X C30T/C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3

## Description for FP-Sigma, FP-X

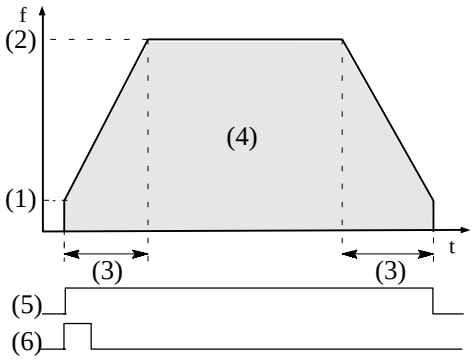
Use the following predefined DUT: **F171_PulseOutput_Trapezoidal_DUT**.

The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed
- Acceleration/deceleration time
- Target value

- Pulse stop

Pulse output characteristics




1. Initial and final speed
2. Target speed
3. Acceleration/deceleration time
4. Target value
5. Pulse output control flag
6. Execution condition

The pulse output frequency changes according to the specified acceleration/deceleration time.The difference between target and initial speed determines the slope of the ramps.

## General programming information

- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- FP-X: When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: The high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: Executing the circular interpolation control instruction **F176_PulseOutput_Center** sets the circular interpolation control flag (**sys_blsCircularInterpolationActive**) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.

- FPΣ: Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- FP-X: Set "Pulse output" for the desired channel in the system registers.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

## Description for FP0R

Use the following predefined DUT: **F171_PulseOutput_Trapezoidal_Type0_DUT** (maximum speed = first target speed) or **F171_PulseOutput_Trapezoidal_Type1_DUT** (maximum speed = 50kHz).

The target speed can be changed during pulse output.

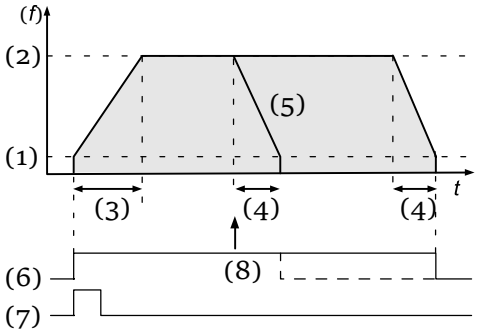
Two control methods are available:

- Type 0: The speed can be changed within the range of the target speed specified first.
- Type 1: The speed can be changed within the range of the maximum speed (50kHz).

The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed
- Acceleration time
- Deceleration time
- Target value

Pulse output characteristics




1. Initial and final speed
2. Target speed
3. Acceleration time

4. Deceleration time
5. Target value
6. Pulse output control flag
7. Execution condition
8. Decelerated stop request

- Type 0:

  The difference between target speed and initial speed determines the slope of the acceleration ramp.The difference between target speed and final speed determines the slope of the deceleration ramp.

- Type 1:

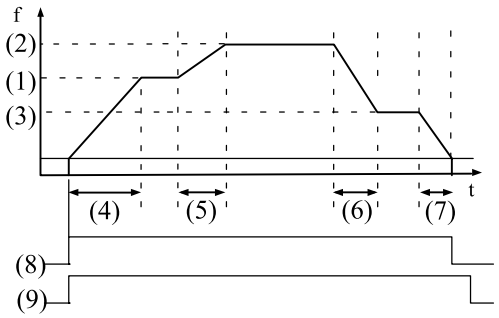
  The difference between the maximum speed of 50kHz and the initial speed determines the slope of the acceleration ramp.The difference between the maximum speed of 50kHz and the final speed determines the slope of the deceleration ramp.

Pulses are output using a duty of 25%.With the pulse output method "pulse/direction", pulses are output approx. 300μs after the direction signal has been output; the motor driver characteristics are simultaneously taken into consideration.

Decelerated stop

To perform a decelerated stop, set bit 5 of the data register storing the pulse output control code from FALSE to TRUE (e.g. `MOVE(16#120, sys_wHscOrPulseControlCode);`).When a decelerated stop is requested during acceleration, deceleration is performed with the same slope as deceleration from the target speed.

Changing the target speed during pulse output




Type 1: The speed can be changed within the range of the maximum speed (50kHz).

1. Target speed
2. 1st change of target speed
3. 2nd change of target speed
4. Acceleration time
5. Acceleration
6. Deceleration
7. Deceleration time

8. Pulse output control flag
9. Execution condition

To change the speed, keep the execution condition TRUE.

- Type 0:
  If a value larger than the target speed at start-up is specified, it will be corrected to the target speed at start-up.
- Type 1:
  If the target speed is set to a value larger than 50kHz, it will be corrected to 50kHz.

If the elapsed value crosses over the acceleration forbidden area starting position (e.g. **sys_diPulseChannel0AccelerationForbiddenAreaStartingPosition**) during acceleration, acceleration cannot be performed.

- The deceleration speed cannot be lower than the corrected final speed.

### General programming information

- As soon as you begin editing a program online (i.e., in RUN mode) using this instruction, pulse output will stop.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- The instruction cannot be started when a decelerated stop has been requested.
- To restart after stopping the operation, turn the execution condition to FALSE and then to TRUE again.
- The execution of the instruction is faster the second time it is started if the positioning parameters remain unchanged. Changing the setting of the output operation (pulse output or calculation only) does not effect this behavior.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

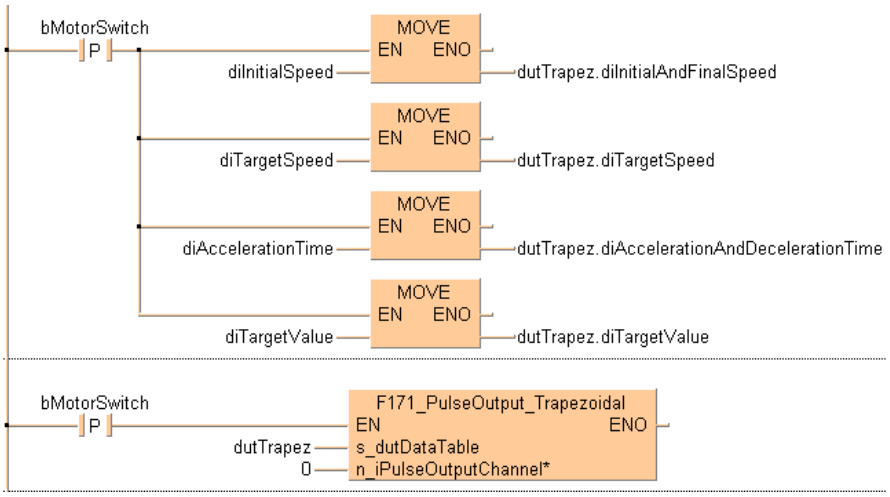
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | X0_bMotorSwitch | BOOL | FALSE | at X0 |
| 1 | VAR | bMotorSwitch | BOOL | FALSE | |
| 2 | VAR | dutTrapez | F171_PulseOutput_Trapezoidal_DUT | dwControlCode := 16#1100 | Control code:<br>Digit 3: 1=Duty ratio 25%<br>Digit 2: 1=Frequency range 48Hz-100kHz<br>Digit 1: 0=Relative value control<br>Digit 0: 0=CW/CCW |
| 3 | VAR | diInitialSpeed | DINT | 100 | |
| 4 | VAR | diTargetSpeed | DINT | 2000 | |
| 5 | VAR | diAccelerationTime | DINT | 300 | |
| 6 | VAR | diTargetValue | DINT | 10000 | |

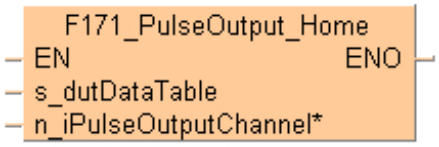
## LD body



# F171_PulseOutput_Home

Home return

This instruction performs a home return according to the parameters in the specified DUT.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**s_dutDataTable (F171_PulseOutput_Home_DUT)**

Starting address of area containing the data table

**n_iPulseOutputChannel*** (decimal constant)

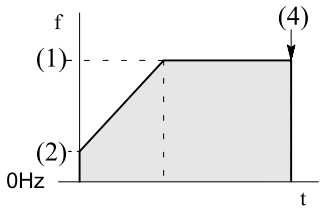
Pulse output channel:
FP-Σ: 0,2
FP-X R: 0,1
FP-XC14T: 0,1,2
FP-X C30T/C60T: 0,1,2,3

## Remarks

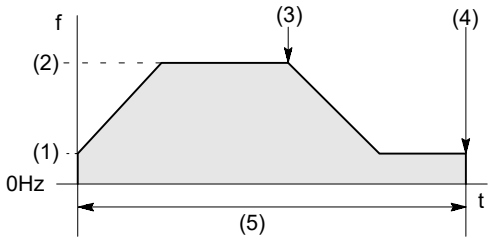
After a drive system has been switched on, there is a difference between the internal position value (elapsed value) and the mechanical position of the axis; this difference cannot be predetermined. The internal value must be synchronized with the actual position value of the axis. This is done by means of a home return, during which a position value is registered at a known reference point (home).During execution of a home return instruction, pulses are continuously output until the home input is enabled. The I/O allocation is determined by the channel used.To decelerate movement when near the home position, designate a near home input and set bit 4 of the special data register storing the pulse output control code (sys_wHscOrPulseControlCode) to TRUE and back to FALSE again.The deviation counter clear output can be set to TRUE when home return has been completed.The value in the elapsed value area during a home return differs from the current value. When the return is completed, the elapsed value changes to 0.

Select one of two different operation modes:

- Type 1: The home input is effective regardless of whether or not there is a near home input, whether deceleration is taking place, or whether deceleration has been completed.
  - Without near home input:

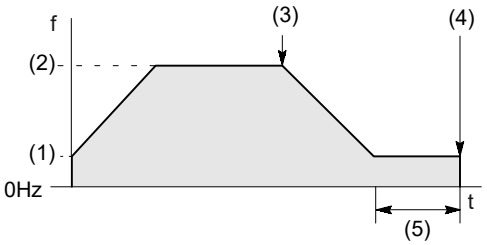



– With near home input:




(1) Initial and final speed
(2) Target speed
(3) Near home input: TRUE
(4) Home input: TRUE
(5) Home input is effective at any time.

- Type 2: The home input is effective only after deceleration (started by near home input) has been completed.




(1) Initial and final speed
(2) Target speed
(3) Near home input: TRUE
(4) Home input: TRUE
(5) Home input is effective only after deceleration

Use the following predefined DUT:

F171_PulseOutput_Home_DUT

The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed
- Acceleration/deceleration time
- Deviation counter clear signal

Pulse output characteristics

- The pulse output frequency changes according to the specified acceleration/deceleration time.
- The difference between target and initial speed determines the slope of the ramps.

General programming information

- Even when home input has occurred, executing this instruction causes pulse output to begin.
- If the near home input is enabled while acceleration is in progress, deceleration will start.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- The deviation counter clear signal is allocated to dedicated output numbers specific to each PLC type.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.
- FP-X: When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: The high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- FP-X: Set "Pulse output" for the desired channel in the system registers.
- FPΣ: Executing the circular interpolation control instruction **F176_PulseOutput_Center** sets the circular interpolation control flag (**sys_blsCircularInterpolationActive**) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

### Deviation counter clear outputs and home inputs for FPΣ

| Channel no. | Deviation counter clear output | Home input |
|---|---|---|
| 0 | Y2 | X2 |
| 2 | Y5 | X5 |

### Deviation counter clear outputs and home inputs for FP-X C14R, C30/60R

The pulse output function is only available if the pulse I/O cassette (AFPX-PLS) has been installed.

| Channel no. | Deviation counter clear output | Home input |
|---|---|---|
| 0 | | |
| (cassette mounting part 1) | Y102 | X102 |
| 1 | | |
| (cassette mounting part 2) | | |
| (C30/60R only) | Y202 | X202 |

### Deviation counter clear outputs and home inputs for FP-X C14T, C30/60T

| Channel no. | Deviation counter clear output | Home input |
|---|---|---|
| 0 | Y4 or Y8 | X4 |
| 1 | Y5 or Y9 | X5 |
| 2 | – | X6 |
| 3 (C30/60R only) | – | X7 |

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global Variables

| | Class | Identifier | FP Address | IEC Address | Type |
|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | X0_bMotorSwitch | X0 | %IX0.0 | BOOL |

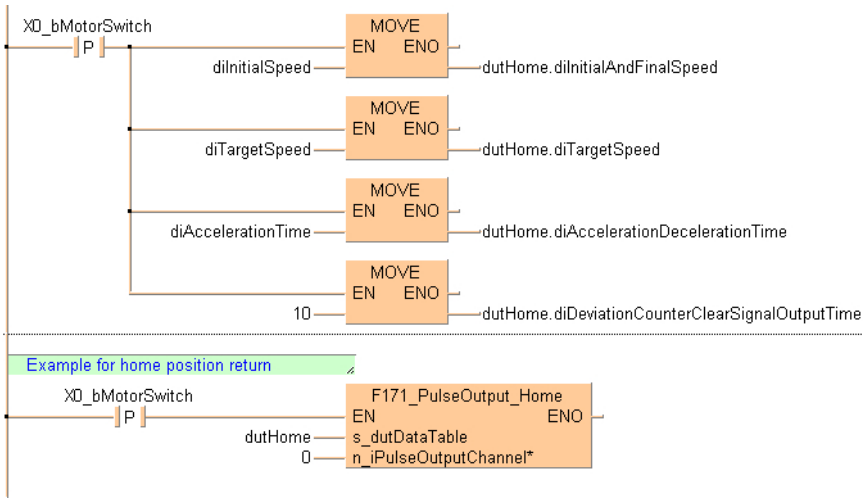
DUT
The DUT F171_PulseOutput_Home_DUT is predefined in the FP Library.

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | X0_bMotorSwitch | BOOL | FALSE | at X0 |
| 1 | VAR | dutHome | F171_PulseOutput_Home_DUT | dwControlCode := 16#1125 | Control code: |
| 2 | VAR | diInitialSpeed | DINT | 100 | Digit 3: 1= Duty ratio 25% |
| 3 | VAR | diTargetSpeed | DINT | 2000 | Digit 2: 1= Frequency range 48Hz-100kHz |
| 4 | VAR | diAccelerationTime | DINT | 300 | Digit 1: 2 =Operation mode type 1 |
| 5 | VAR | | | | Ditig 0: 5= CCW + deviation counter clear signal |

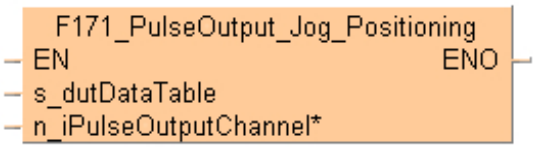
## LD body



# F171_PulseOutput_Jog_Positioning

JOG operation and positioning

The specified number of pulses is output after the position control trigger input has turned to TRUE. A deceleration is performed before the target value is reached and pulse output stops.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**s_DUT_DataTable** (DUT)

Starting address of area containing the data table
**F171_PulseOutput_Jog_Positioning_Type0_DUT** or
**F171_PulseOutput_Jog_Positioning_Type1_DUT**

**n_iPulseOutputChannel** (decimal constant)

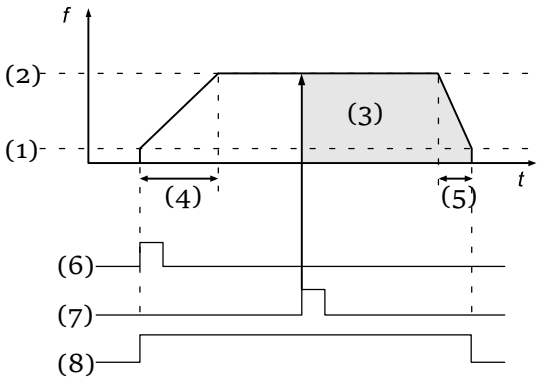
Pulse output channel:0–3

## Remarks

Select one of two different operation modes:

- Type 0: The speed can be changed within the range of the specified target speed.
- Type 1: The target speed can be changed once when the position control trigger input turns to TRUE.

Pulse output characteristics




(1) Initial and final speed
(2) Target speed
(3) Target value
(4) Acceleration time
(5) Deceleration time
(6) Execution condition
(7) Position control trigger input
(8) Pulse output control flag

- The pulse output frequency changes according to the specified acceleration time and the specified deceleration time.
- The difference between target speed and initial speed determines the slope of the acceleration ramp.
- The difference between target speed and final speed determines the slope of the deceleration ramp.
- After the position control trigger input has turned to TRUE, pulse output continues, then decelerates and stops when the target value is reached.

### Stopping pulse output

Pulse output can be stopped by one of the following operations:

- Turning the position control trigger to TRUE (pulse output continues until the target value has been reached and deceleration has completed): The position control trigger can be started by turning a position control trigger input to TRUE or by setting bit 6 of the data register storing the pulse output control code from FALSE to TRUE (e.g. `MOVE(16#140, sys_wHscOrPulseControlCode);`).
- Requesting a decelerated stop: To perform a decelerated stop, set bit 5 of the data register storing the pulse output control code from FALSE to TRUE (e.g. `MOVE(16#120, sys_wHscOrPulseControlCode);`).When a decelerated stop is requested during acceleration, deceleration is performed with the same slope as deceleration from the target speed.
- Executing an emergency stop: To perform an emergency stop, set bit 3 of the data register storing the pulse output control code from FALSE to TRUE (e.g. `MOVE(16#108, sys_wHscOrPulseControlCode);`).

note

When stopping, disable all pulse output functions for the channel used in the program.

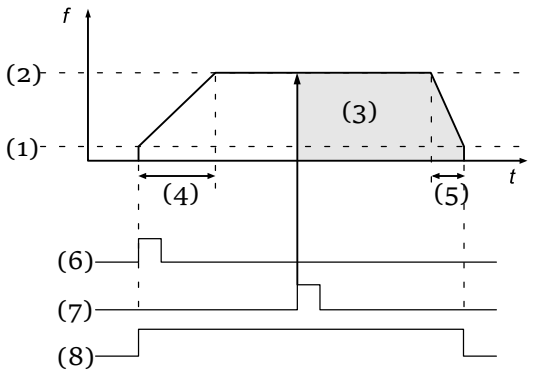
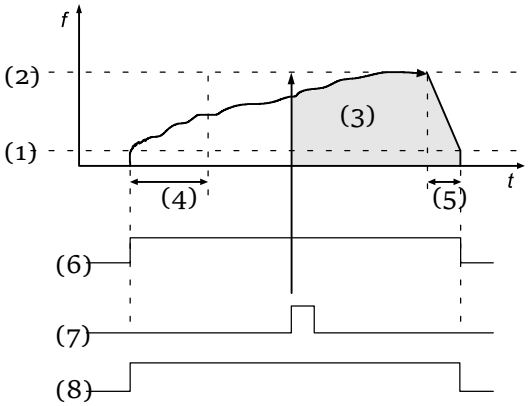
## JOG Operation Type 0

Use the following predefined DUT: F171_PulseOutput_Jog_Positioning_Type0_DUT

The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed
- Acceleration time
- Deceleration time
- Target value

The target speed can be changed during pulse output.Changing the target speed during pulse output

| Without changing the target speed: | With changing the target speed: |
|---|---|
|  |  |



1. Initial and final speed
2. Target speed
3. Target value
4. Acceleration time
5. Deceleration time
6. Execution condition
7. Position control trigger input
8. Pulse output control flag

- To change the speed, keep the execution condition TRUE.

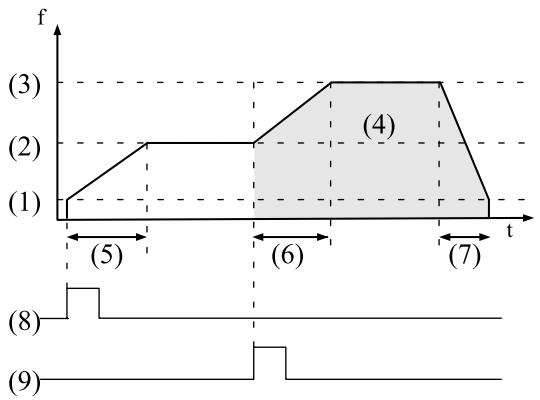
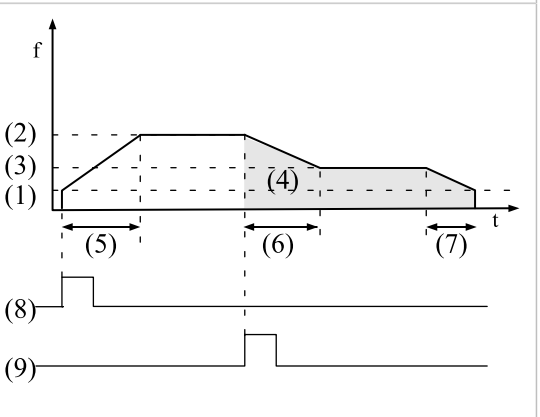
- If the target speed is set to a value larger than 50kHz, it will be corrected to 50kHz.
- If the elapsed value crosses over the acceleration forbidden area starting position (e.g. **sys_diPulseChannel0AccelerationForbiddenAreaStartingPosition**) during acceleration, acceleration cannot be performed.
- The deceleration speed cannot be lower than the corrected final speed.
- Changing the target speed is not possible if the instruction is executed in an interrupt program.

### JOG Operation Type 1

Use the following predefined DUT: F171_PulseOutput_Jog_Positioning_Type1_DUT

The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed1
- Acceleration time
- Target speed2
- Change time
- Deceleration time
- Target value

| Target speed 1< target speed 2: | Target speed 1> target speed 2: |
|---|---|
|  |  |





1. Initial and final speed
2. Target speed
3. Target speed
4. Target value
5. Acceleration time
6. Change time
7. Deceleration time
8. Execution condition

9. Position control trigger input

After the position control trigger input has turned to TRUE, the pulse output frequency will change using the change time to accelerate or decelerate to target speed 2. Further target speed changes are not possible. The position control trigger input will be disregarded if it is turned on during acceleration.

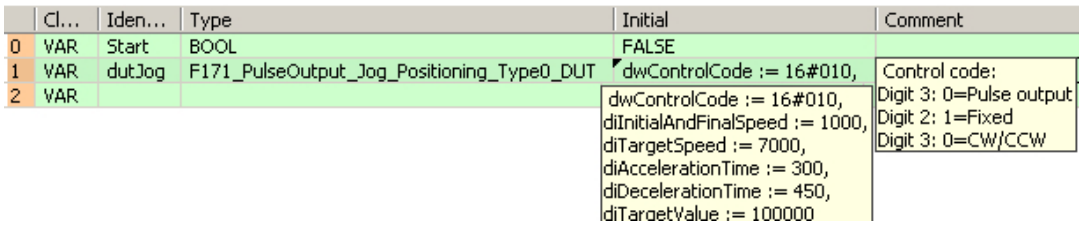
## General programming information

- As soon as you begin editing a program online (i.e., in RUN mode) using this instruction, pulse output will stop.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- Set the position control trigger input (X0, X1, X2, X3) in system register 402.
- For the position control trigger input, only the rising edge (TRUE) is detected.
- The instruction cannot be started when a decelerated stop has been requested.
- To restart after stopping the operation, turn the execution condition to FALSE and then to TRUE again.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

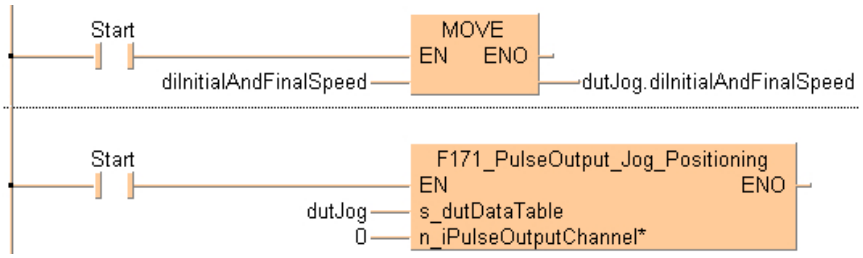
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.



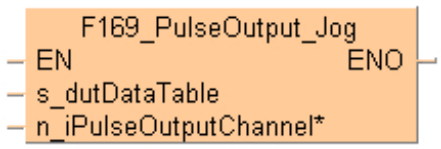
### LD body


Start
MOVE
EN ENO
dilnitialAndFinalSpeed
dutJog.dilnitialAndFinalSpeed
Start
F171_PulseOutput_Jog_Positioning
EN ENO
dutJog
s_dutDataTable
0
n_iPulseOutputChannel*

# F169_PulseOutput_Jog

JOG operation

This instruction is used for JOG operation.The specified number of pulses is output after the position control trigger input has turned to TRUE. A deceleration is performed before the target value is reached and pulse output stops.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**s_dutDataTable** (F169_PulseOutput_Jog_DUT)

Starting address of area containing the data table

**n_iPulseOutputChannel** (INT)

Pulse output: 0 or 1

## Remarks

- Use the following predefined DUT: **F169_PulseOutput_Jog_DUT**
- The following parameters can be specified in the DUT:
  - Control code
  - Speed
- Pulse output characteristics

  The frequency and the duty can be changed in each scan. (The change becomes effective with the next pulse output.)

## General programming information

- As soon as you begin editing a program online (i.e., in RUN mode) using this instruction, pulse output will stop.
- Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- The high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) are assigned to the

same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_blsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.

- Pulse output stops when the upper limit of the internal elapsed value is exceeded if rotation is in one direction only. As a countermeasure, reset the elapsed value to 0 before executing this instruction. Pulse output does not stop when the FP0R is used in FP0 compatibility mode because the data range for the elapsed value is a signed 32-bit value.
- When using incremental counting, pulse output stops when the elapsed value exceeds 2147483647.
- When using decremental counting, pulse output stops when the elapsed value exceeds -2147483648.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

### Running the FP0R in FP0 compatibility mode

To run the FP0R in FP0 compatibility mode, you can download an FP0 program to the FP0R. Please note the following restrictions:

- The FP0R supports signed 32-bit data for elapsed value and target value; the FP0 supports signed 24-bit data. In FP0 compatibility mode, counting and pulse output continue even if data exceeds the FP0 range.
- The duty ratio is always 25% regardless of the settings in the instructions. With the pulse output method "pulse/direction", pulses are output approx. 300μs after the direction signal has been output; the motor driver characteristics are simultaneously taken into consideration.
- The FP0R does not support the "no counting" setting. Instead, incremental counting is performed with the FP0 pulse output instructions set to "no counting".
- The maximum pulse output frequency is 10000Hz.
- Make sure the pulse output instruction does not use an output that is also being used as a normal output.
- An FP0 program can only run in FP0 compatibility mode, if the PLC types (C10, C14, C16, C32, and T32) match exactly. FP0 compatibility mode is not available for the F32 type FP0R.

### Outputs and system variables for FP0, FP-e

Channel and pulse output numbers

| Channel no. | Pulse output | Pulse output method |
|---|---|---|
| **0** | Y0 | Pulse |
| | Y2 | Direction |
| **1** | Y1 | Pulse |
| | Y3 | Direction |

System variables for memory areas used. Values in parentheses are valid for FP0 T32.

| **Description** | | **System variable** |
|---|---|---|
| Pulse output: control flag for channel | 0 | **sys_blsPulseChannel0Active** |
| | 1 | **sys_blsPulseChannel1Active** |
| Pulse output: elapsed value for channel | 0 | **sys_diPulseChannel0ElapsedValue** |
| | 1 | **sys_diPulseChannel1ElapsedValue** |
| Pulse output: target value for channel | 0 | **sys_diPulseChannel0TargetValue** |
| | 1 | **sys_diPulseChannel1TargetValue** |
| High-speed counter or pulse output controlcode | | **sys_wHscOrPulseControlCode** |

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global Variables

| | Class | Identifier | FP Address | IEC Address | Type |
|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | X0_bMotorSwitch | X0 | %IX0.0 | BOOL |

DUT
The DUT F169_PulseOutput_Jog_DUT is predefined in the FP Library.

### POU header

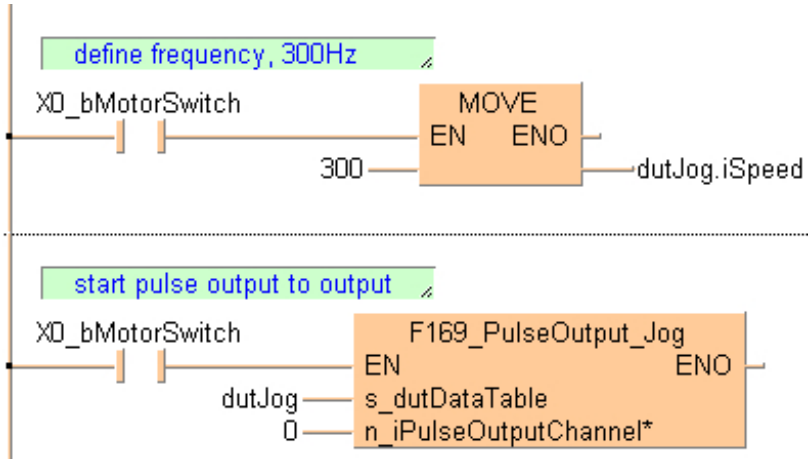
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | X0_bMotorSwitch | BOOL | FALSE | |
| 1 | VAR | dut_Jog | F169_PulseOutput_Jog_DUT | wControlCode := 16#110, | Digit 2: 1=Duty 10%<br>Digit 1: 1=Incremental counting<br>Digit 0: 0=No direction output<br>Speed: 300 Hz |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |

### POU body

The comment fields explain the function of this example.

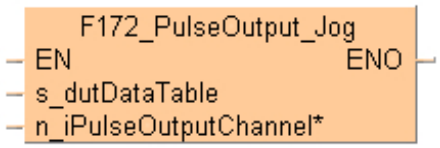
### LD body


define frequency, 300Hz
X0_bMotorSwitch
MOVE
EN
ENO
300
dutJog.iSpeed
start pulse output to output
X0_bMotorSwitch
F169_PulseOutput_Jog
EN
ENO
dutJog
s_dutDataTable
0
n_iPulseOutputChannel*

# F172_PulseOutput_Jog

JOG operation

This instruction is used for JOG operation.The specified number of pulses is output after the position control trigger input has turned to TRUE. A deceleration is performed before the target value is reached and pulse output stops.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**s_dutDataTable**

Starting address of area containing the data table

- FP-Σ, FP-X:

  **F172_PulseOutput_Jog_Type0_DUT_0**

  **F172_PulseOutput_Jog_Type1_DUT_0**

- FP0R:

  **F172_PulseOutput_Jog_Type0_DUT_1**

  **F172_PulseOutput_Jog_Type1_DUT_1**

**n_iPulseOutputChannel*** (decimal constant)

Pulse output channel:
FP-XH C30 T/P: 0–3
FP-XH C60 T/P: 0–5
FP-Σ: 0,2
FP-X R: 0,1
FP-XC14T: 0,1,2
FP-X C30T/C60T: 0,1,2,3
FP0R: 0,1,2,3

## Description for FP-Sigma, FP-X

Use the following predefined DUT: **F172_PulseOutput_Jog_Type0_DUT_0** (Mode with no target value) or**F172_PulseOutput_Jog_Type1_DUT_0** (Target value match stop mode)

The following parameters can be specified in the DUT:

- Control code
- Frequency

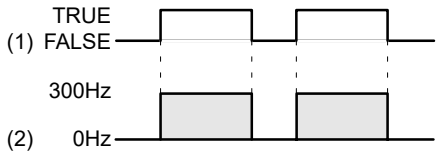
- Target value

Pulse output characteristics

The frequency and the target value can be changed in each scan. The control code, however, cannot be changed during execution of the instruction.Select one of two different operation modes:

- Mode with no target value (type 0):

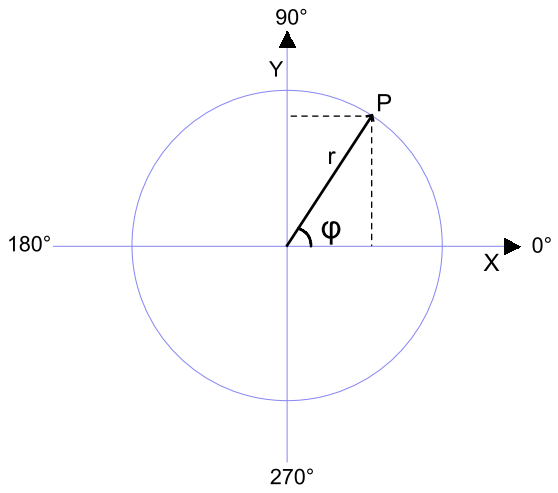
  Pulses are output in accordance with the conditions set in the DUT as long as the execution condition is TRUE.




  (1) Execution condition
  (2) CW pulse output

- Target value match stop mode (type 1):

  Output stops when the target value is reached. Set this mode in the control code, and specify the target value (an absolute value) in the DUT. (FPΣV1.4 or higher, FP-X)




  (1) Execution condition
  (2) CW pulse output
  (3) Target value reached (pulse output stops)

General programming information

NOTICE

As soon as you begin editing a program online (i.e., in RUN mode) using this instruction, pulse output will stop.

If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.

- FP-X: When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: The high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: Executing the circular interpolation control instruction **F176_PulseOutput_Center** sets the circular interpolation control flag (**sys_bIsCircularInterpolationActive**) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed.
- FPΣ: Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- FP-X: Set "Pulse output" for the desired channel in the system registers.
- If the execution of the instruction is started with an invalid frequency value, an operation error occurs. If the frequency is changed to an invalid value during execution of the instruction, the frequency output will be adjusted to either the minimum or the maximum value of the permissible range.
- Changing the control code during execution of the instruction will have no effect.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

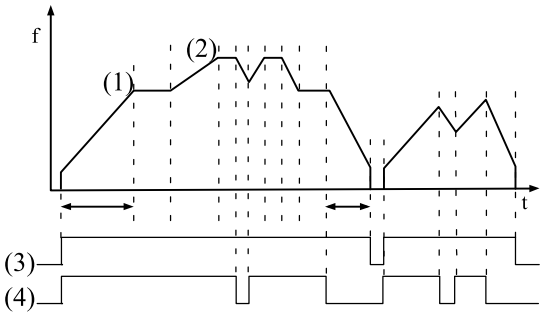
## Description for FP0R

Use the following predefined DUT: **F172_PulseOutput_Jog_Type0_DUT_1** (Mode with no target value) or **F172_PulseOutput_Jog_Type1_DUT_1** (Target value match stop mode)

The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed
- Acceleration time
- Deceleration time
- Target value

Pulse output characteristics



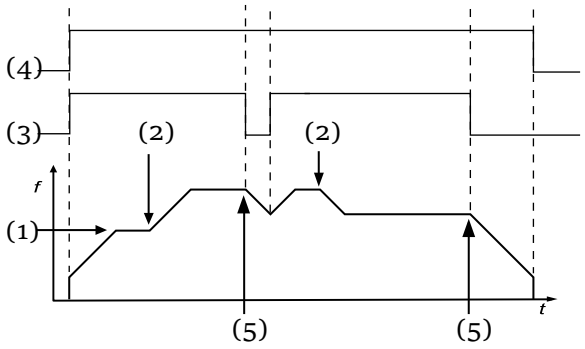

(1) Target speed1
(2) Target speed2
(3) Pulse output control flag
(4) Execution condition

- The pulse output frequency changes according to the specified acceleration time and the specified deceleration time.
- The difference between the maximum speed of 50kHz and the initial speed determines the slope of the acceleration ramp.
- The difference between the maximum speed of 50kHz and the final speed determines the slope of the deceleration ramp.
- When the execution condition turns to FALSE after starting the instruction, a decelerated stop is performed.
- When the execution condition turns to TRUE during deceleration, acceleration is performed again.
- The target speed can be changed during pulse output.
- Pulses are output using a duty of 25%.
- With the pulse output method "pulse/direction", pulses are output approx. 300μs after the direction signal has been output; the motor driver characteristics are simultaneously taken into consideration.
- When a decelerated stop is requested during acceleration, deceleration is performed with the same slope as deceleration from the target speed.
- Acceleration time and deceleration time have priority over initial speed and final speed. This means that the values for acceleration time and deceleration time will not be changed whereas the values for initial speed and final speed may be corrected by the pulse output instruction to enable acceleration and deceleration within the specified time. The modified values are written to data registers which can be accessed using the system variables **sys_iPulseChannelxCorrectedInitialSpeed** and **sys_iPulseChannelxCorrectedFinalSpeed** (where x=channel number

Select one of two different operation modes:

- Mode with no target value (type 0):

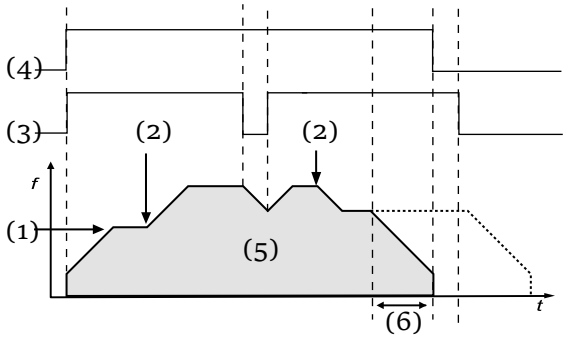
  Pulses are output in accordance with the conditions set in the DUT as long as the execution condition is TRUE.A decelerated stop begins whenever the execution condition is FALSE.




(1) Initial and final speed
(2) Change of target speed
(3) Execution condition
(4) Pulse output control flag
(5) Decelerated stop

- Target value match stop mode (type 1):

  Output stops when the target value is reached. Set this mode in the control code, and specify the target value (an absolute value) in the DUT.A decelerated stop is performed when the target value has been reached. Deceleration is performed within the specified deceleration time.




(1) Initial and final speed
(2) Change of target speed
(3) Execution condition
(4) Pulse output control flag
(5) Target value
(6) Deceleration time

Changing the target speed during pulse output

- If the elapsed value crosses over the acceleration forbidden area starting position (e.g. **sys_diPulseChannel0AccelerationForbiddenAreaStartingPosition**) during acceleration, acceleration cannot be performed.
- The deceleration speed cannot be lower than the corrected final speed.

General programming information

NOTICE

As soon as you begin editing a program online (i.e., in RUN mode) using this instruction, pulse output will stop.

If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.

- When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- Changing the control code during execution of the instruction will have no effect.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

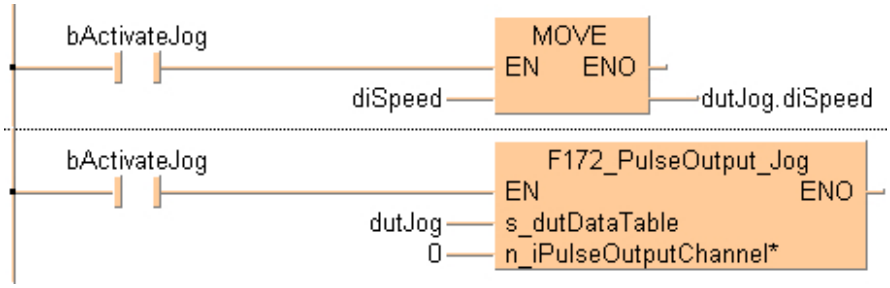
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.  The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | bActivateJog | BOOL | FALSE | |
| 1 | VAR | dutJog | F172_PulseOutput_Jog_Type0_DUT_0 | dwControlCode := 16#1110 | Control code:<br>Digit 3: 1=Duty ratio 25%<br>Digit 2: 1=Frequency range 48Hz-100kHz<br>Digit 1: 1=Incremental counting<br>Digit 0: 0=CW |
| 2 | VAR | diSpeed | DINT | 300 | |
| 3 | VAR | | | | |

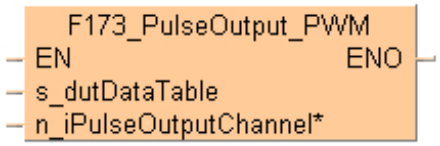
### LD body



# F173_PulseOutput_PWM

Pulse output instruction with channel specification (PWM output)

This instruction delivers a pulse width modulated output signal according to the specified DUT. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**s_dutDataTable (F173_PulseOutput_PWM_DUT)**

Starting address of area containing the data table

**n_iPulseOutputChannel*** (decimal constant)

Pulse output channel:
FP-XH C30 T/P: 0–3
FP-XH C60 T/P: 0–5
FPΣ: 0,2
FP-X, FP-XH Relay: 0,1
FP-X, FP-XH 16k Transistor: 0–2
FP0R, FP0H, FP-X, FP-XH 32 Transistor: 0–3
FP-XH 32k Transistor: 0–5

## Remarks

Use the following predefined DUT: **F173_PulseOutput_PWM_DUT**

The following parameters can be specified in the DUT:

- Approximate frequency
- Duty ratio (for pulse duration and period)

  The ratio between the pulse duration and the period of a rectangular waveform. For a pulse train in which the pulse duration is 1μs and the pulse period is 4μs, the duty ratio is 0.25 or 25%.

## General programming information

- The duty ratio, particularly when it is close to the minimum or maximum value, may differ from the specified duty ratio, depending on the load voltage and the load current.

- The duty ratio can be changed for each scan.
- The frequency constant K cannot be changed during execution of the instruction. If it is changed, it will have no effect on the frequency but on the resolution of the duty ratio.
- If the duty ratio is changed to a value outside the permissible range while the instruction is being executed, the duty ratio is adjusted to the maximum value. When execution of the instruction begins, an operation error is displayed.
- If the frequency is changed to a value outside the permissible range while the instruction is being executed, the resolution is adjusted to 100. When execution of the instruction begins, no operation error is displayed.
- If the duty is changed to 100% or higher while the instruction is being executed, the frequency is adjusted to the maximum value at the specified resolution. When execution of the instruction begins, no operation error is displayed.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- FP-X, FP0R: When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: The high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: Executing the circular interpolation control instruction **F176_PulseOutput_Center** sets the circular interpolation control flag (**sys_bIsCircularInterpolationActive**) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed.
- FPΣ: Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- FP-X, FP0R: Set "PWM output" for the desired channel in the system registers.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global Variables

| | Class | Identifier | FP A... | IEC Addr... | Type | Initial |
|---|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | X6_bEnablePWM | X6 | %IX0.6 | BOOL | FALSE |

### DUT

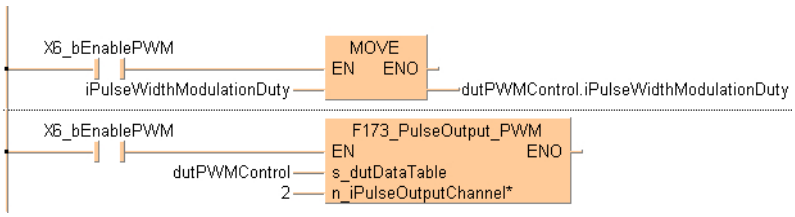
The DUT **F173_PulseOutput_PWM_DUT** is predefined in the FP Library.

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | X6_bEnablePWM | BOOL | FALSE | |
| 1 | VAR | dutPWMControl | F173_PulseOutput_PWM_DUT | iFrequencyValue := 1 | iFrequencyValue := 1: f=2.0 Hz, T=502.5 ms; |
| 2 | VAR | iPulseWidthModulationDuty | INT | 500 | 500 = 50% duty |

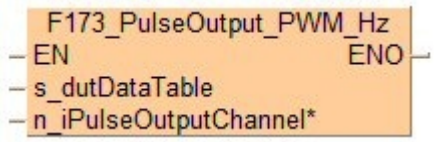
### LD body



# F173_PulseOutput_PWM_Hz

Pulse output instruction with channel specification (PWM output)

This instruction delivers a pulse width modulated output signal according to the specified DUT. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**s_dutDataTable (F173_PulseOutput_PWM_Hz_DUT)**

Starting address of area containing the data table

**n_iPulseOutputChannel*** (decimal constant)

Pulse output channel:
FP-XH C30 T/P: 0–3
FP-XH C60 T/P: 0–5
FPΣ: 0,2
FP-X, FP-XH Relay: 0,1
FP-X, FP-XH 16k Transistor: 0–2
FP0R, FP0H, FP-X, FP-XH 32 Transistor: 0–3
FP-XH 32k Transistor: 0–5

## Remarks

Use the following predefined DUT: **F173_PulseOutput_PWM_Hz_DUT**

The following parameters can be specified in the DUT:

- Approximate frequency
- Duty ratio (for pulse duration and period)

  The ratio between the pulse duration and the period of a rectangular waveform. For a pulse train in which the pulse duration is 1μs and the pulse period is 4μs, the duty ratio is 0.25 or 25%.

## General programming information

- The duty ratio, particularly when it is close to the minimum or maximum value, may differ from the specified duty ratio, depending on the load voltage and the load current.

- The duty ratio can be changed for each scan.
- The frequency constant K cannot be changed during execution of the instruction. If it is changed, it will have no effect on the frequency but on the resolution of the duty ratio.
- If the duty ratio is changed to a value outside the permissible range while the instruction is being executed, the duty ratio is adjusted to the maximum value. When execution of the instruction begins, an operation error is displayed.
- If the frequency is changed to a value outside the permissible range while the instruction is being executed, the resolution is adjusted to 100. When execution of the instruction begins, no operation error is displayed.
- If the duty is changed to 100% or higher while the instruction is being executed, the frequency is adjusted to the maximum value at the specified resolution. When execution of the instruction begins, no operation error is displayed.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- FP0H, FP0R, FP-XH, FP-X: When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_blsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- FP0H, FP0R, FP-XH, FP-X: Set “PWM output” for the desired channel in the system registers.

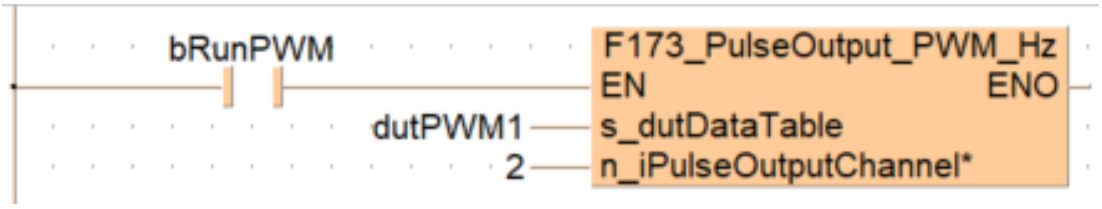
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | dutPWM1 | F173_PulseOutput_PWM_Hz_DUT | diFrequency_Hz:=1000,iDuty:=50 |
| 1 | VAR | bRunPWM | BOOL | FALSE |

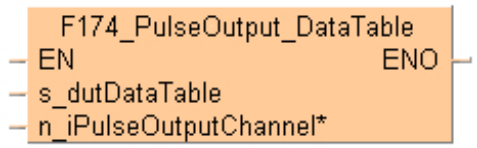
### LD body



# F174_PulseOutput_DataTable

Data table control

This instruction performs rectangular control according to the parameters in the specified DUT with an arbitrary number of different speeds and target values.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**s_dutDataTable (ANY_DUT)**

Starting address of area containing the data table
Sample: **F174_PulseOutput_DataTable_8_Values_DUT**

**n_iPulseOutputChannel*** (decimal constant)

Pulse output channel:
FP-XH C30 T/P: 0–3
FP-XH C60 T/P: 0–5
FPΣ: 0,2
FP-X R: 0,1
FP-XC14T: 0,1,2
FP-X C30T/C60T: 0,1,2,3
FP0R: 0,1,2,3

## Remarks

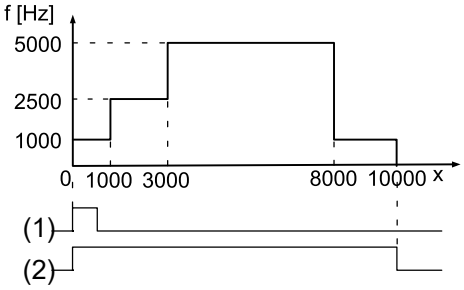
Create your own DUT using the following DUT as a sample:
**F174_PulseOutput_DataTable_8_Values_DUT**

The following parameters can be specified in the DUT:

- Control code
- Frequency 1
- Target value 1
- Frequency 2
- Target value 2
- ...

- Frequency **n**
- Target value **n**
- Pulse stop

Pulse output characteristics




(1) Execution condition
(2) Pulse output control flag

x Elapsed value of high-speed counter (amount of travel)

- Pulses are output at the specified frequency until the target value is reached. Then the frequency changes to the second frequency value and pulse output continues until the second target value is reached, and so forth.
- Pulse output stops when the last target value is reached.
- A frequency of 0 signifies the final frequency and stops pulse output.

General programming information

- FP-X, FP0R: When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: The high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- If the value of the first frequency specified is out of range, an operation error occurs. (If the value of the first frequency is 0, operation stops without any pulses having been output.)
- If the value of the second frequency specified is out of range or 0, pulse output stops.
- If the target value is out of range, the number of pulses output may be different from the specified value.
- FP0R: If one of the target values is not in the direction of movement, pulse output stops.

- FPΣ, FP-X: If one of the target values is not in the direction of movement, counting continues infinitely.
- FPΣ: Executing the circular interpolation control instruction **F176_PulseOutput_Center** sets the circular interpolation control flag (**sys_bIsCircularInterpolationActive**) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed.
- FPΣ: Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- FPΣ, FP-X: If a periodic interrupt or high-speed counter interrupt program is run, or the PLC link function is used at the same time, a frequency of 80kHz or less should be used.
- FP-X: Set "Pulse output" for the desired channel in the system registers.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

Global Variables

| | Class | Identifier | FP Address | IEC Address | Type |
|---|---|---|---|---|---|
| 0 | VAR_GLOBAL | X0_bMotorSwitch | X0 | %IX0.0 | BOOL |

### DUT

The DUT **F174_PulseOutput_DataTable_8_Values_DUT** is predefined in the “FP library” and can be used as a sample.

F174_DUT [DUT]

| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | ControlCode | DWORD | 0 | Highest word fixed to 0 |
| 1 | Frequency1 | DINT | 0 | |
| 2 | TargetValue1 | DINT | 0 | |
| 3 | Frequency2 | DINT | 0 | |
| 4 | TargetValue2 | DINT | 0 | |
| 5 | Frequency3 | DINT | 0 | |
| 6 | TargetValue3 | DINT | 0 | |
| 7 | Frequency4 | DINT | 0 | |
| 8 | TargetValue4 | DINT | 0 | |
| 9 | Termination | DINT | 0 | End of data table |

## POU header

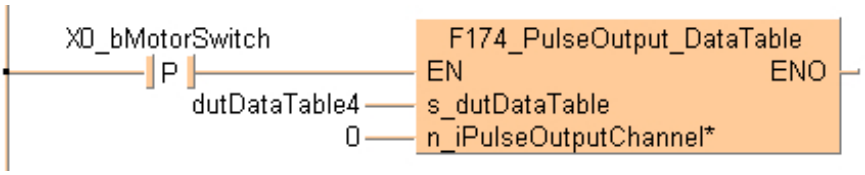
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR_EXTERNAL | x0_bMotorSwitch | BOOL | FALSE | |
| 1 | VAR | dutDataTable4 | F174_DUT | ControlCode := 16#1200, | For ControlCode (16#1200): |

ControlCode := 16#1200,
Frequency1 := 1000,
TargetValue1 := 1000,
Frequency2 := 2500,
TargetValue2 := 2000,
Frequency3 := 5000,
TargetValue3 := 5000

For ControlCode (16#1200):
1 = 25% duty
2 = 191 Hz to 100 kHz
0 = Relative value control
0 = CW (incremental counting)CC

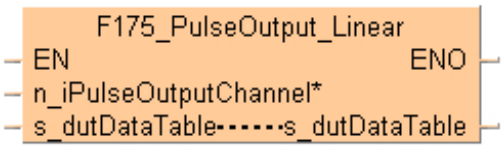
## LD body



# F175_PulseOutput_Linear

Linear interpolation

Pulses are output from two channels in accordance with the parameters in the specified DUT, so that the path to the target position forms a straight line.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**n_iPulseOutputChannel**constant

Pulse output channel:
FP-XH C30 T/P: 0–3
FP-XH C60 T/P: 0–5
FPΣ: 0, 2
FP-X R: 0, 1
FP-XC14T: 0, 1, 2
FP-X C30T/C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
For interpolation, channel 0 and 1 or channel 2 and 3 are used as pairs. You may only specify 0 or 2 (for C14T: 0 only).

Input/output

**s_dutDataTable** (DUT)

Starting address of area containing the data table
FPΣ, FP-X: **F175_PulseOutput_Linear_DUT_0**
FP0R: **F175_PulseOutput_Linear_DUT_1**

## Description for FP-Sigma, FP-X

Use the following predefined DUT: **F175_PulseOutput_Linear_DUT_0**

The following parameters can be specified in the DUT:

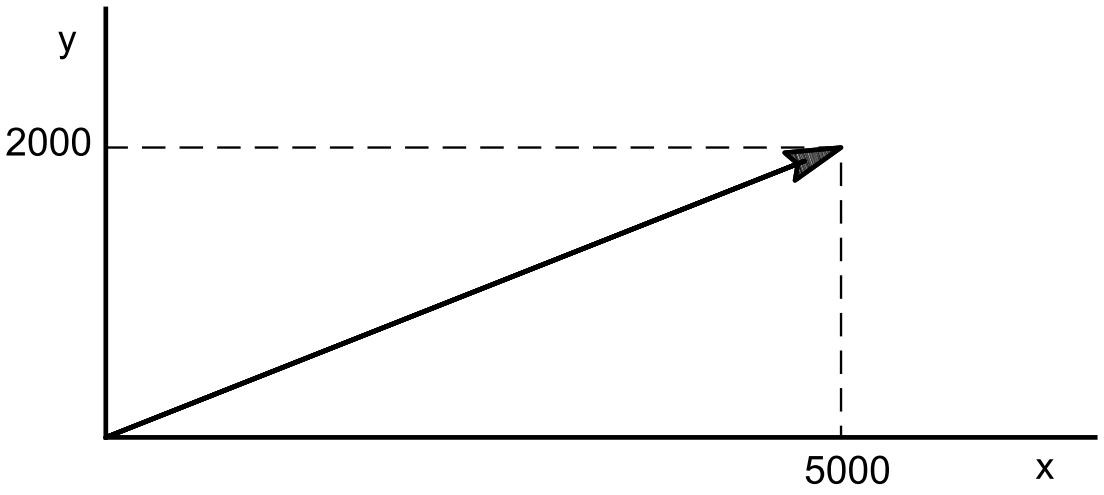
- Control code
- Initial and final speed
- Target speed

- Acceleration/deceleration time
- X-axis target value
- Y-axis target value

The following parameters for each axis are calculated upon execution of the instruction and stored in the operation result area of the DUT.

- X-axis initial and final speed
- X-axis target speed
- Y-axis initial and final speed
- Y-axis target speed
- X-axis frequency range
- Y-axis frequency range
- X-axis number of acceleration/deceleration steps
- Y-axis number of acceleration/deceleration steps

Pulse output characteristics




X-axis target value (channel 0): 5000

Y-axis target value (channel 2) (FP-X: channel 1): 2000

The two axes are controlled so that a linear path is followed to the target position.

## General programming information

- The target value for each axis must be within the range of -8388608–8388607.
  When this instruction is used in combination with other pulse output instructions, e.g. **F171_PulseOutput_Trapezoidal**, the target value in these instructions must be within the same range.

- When using in applications requiring precision, test runs with the actual machine are necessary.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- FP-X: When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: The high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- FPΣ: Executing the circular interpolation control instruction **F176_PulseOutput_Center** sets the circular interpolation control flag (**sys_bIsCircularInterpolationActive**) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed.
- FPΣ: Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- FP-X: Set "Pulse output" for the desired channel in the system registers.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

### Description for FP0R

Use the following predefined DUT: F175_PulseOutput_Linear_DUT_1

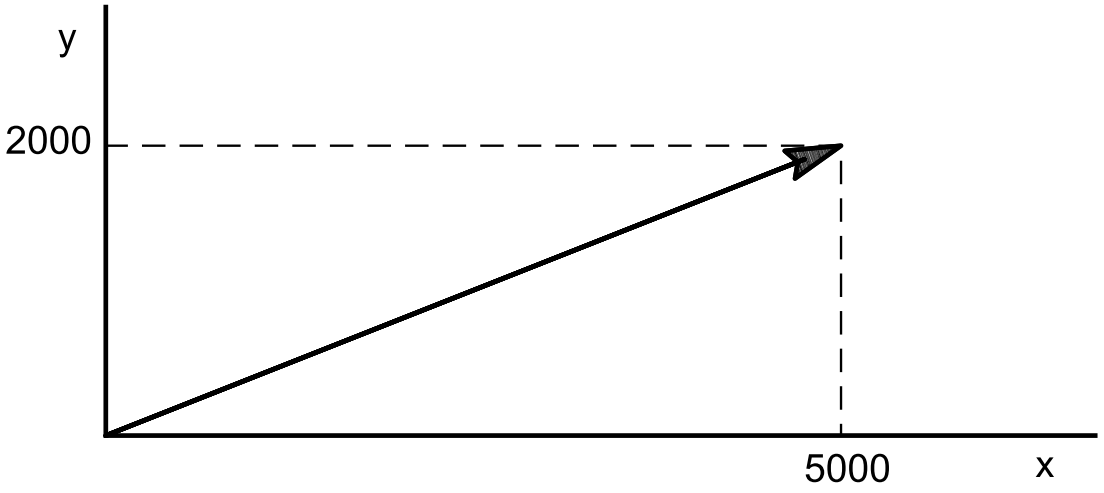
The following parameters can be specified in the DUT:

- Control code
- Initial and final speed
- Target speed
- Acceleration time
- Deceleration time
- X-axis target value
- Y-axis target value

The following parameters for each axis are calculated upon execution of the instruction and stored in the operation result area of the DUT.

- X-axis initial and final speed
- X-axis target speed
- Y-axis initial and final speed
- Y-axis target speed

Pulse output characteristics




X-axis target value (channel 0): 5000

Y-axis target value (channel 1): 2000

Pulses are output from the X-axis (channel 0) and the Y-axis (channel 1), so that the initial speed is 500Hz, the target speed is 5kHz, and the acceleration time and deceleration time is 300ms. The two axes are controlled so that a linear path is followed to the target position.

Pulses are output using a duty of 25%.With the pulse output method "pulse/direction", pulses are output approx. 300μs after the direction signal has been output; the motor driver characteristics are simultaneously taken into consideration.

## General programming information

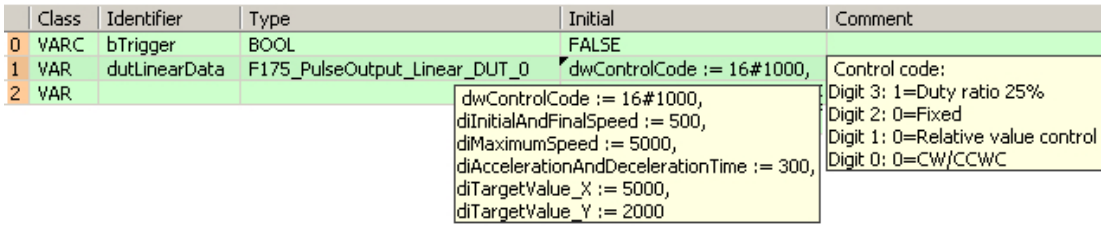
- Pulse output stops when the target value is reached.
- The target value for each axis must be within the range of -8388608–8388607.
  When this instruction is used in combination with other pulse output instructions, e.g. **F171_PulseOutput_Trapezoidal**, the target value in these instructions must be within the same range.
- When using in applications requiring precision, test runs with the actual machine are necessary.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.

- When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

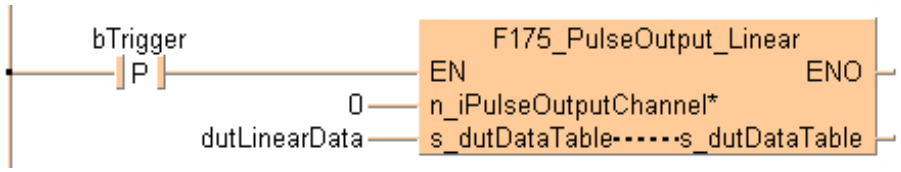
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.



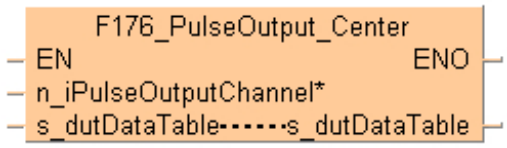

### LD body



# F176_PulseOutput_Center

Circular interpolation (center position)

Pulses are output from two channels in accordance with the parameters in the specified DUT, so that the path to the target position forms an arc.The radius of the circle is calculated by specifying the center position and the end position.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**n_iPulseOutputChannel*** (decimal constant)

Pulse output channel: 0, 2

Input/output

**s_dutDataTable F176_PulseOutput_Center_DUT**

Starting address of area containing the data table

## Remarks

Use the following predefined DUT: **F176_PulseOutput_Center_DUT**

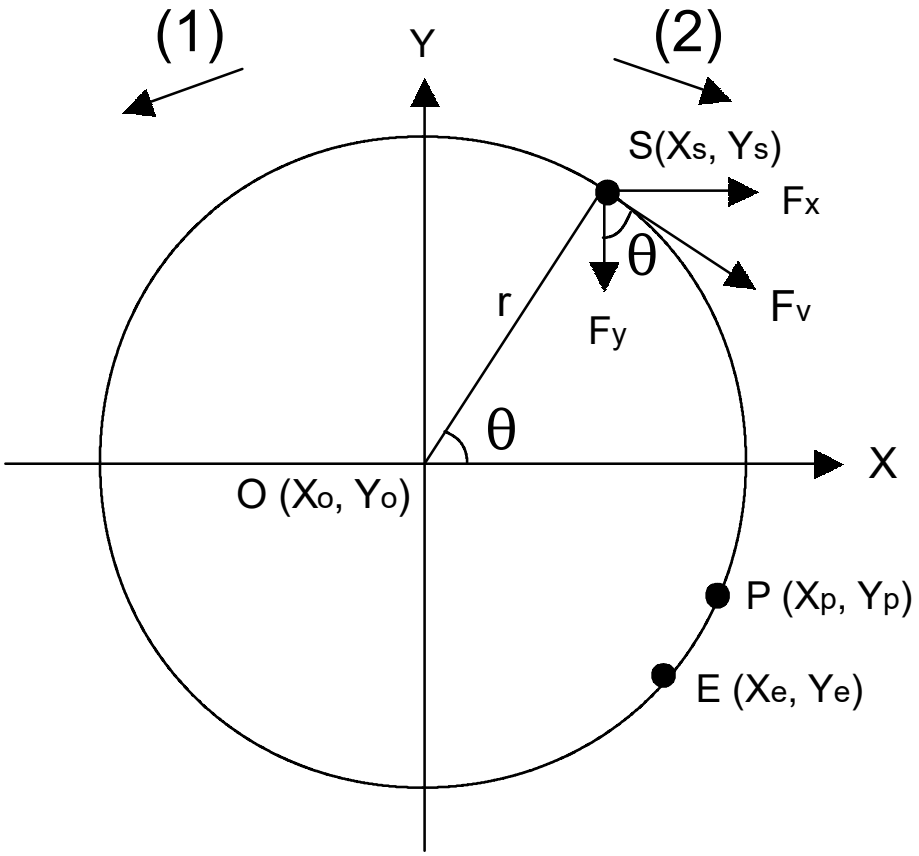
The following parameters can be specified in the DUT:

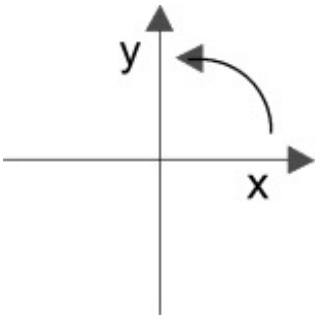
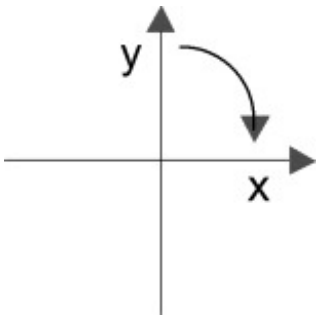
- Control code
- Composite speed
- X-axis target value
- Y-axis target value
- X-axis center value
- Y-axis center value

The following parameters for each axis are calculated upon execution of the instruction and stored in the operation result area of the DUT.

- Radius

Pulse output characteristics




| | | | |
|---|---|---|---|
| (1) | Rotation direction: from channel 0 (x-axis) to channel 2 (y-axis) (for a movement in positive direction on both channels)   | (2) | Rotation direction: from channel 2 (y-axis) to channel 0 ( x-axis) (for a movement in positive direction on both channels)   |
| **$F_v$:** | Composite speed | **O (Xo,Yo):** | Center position |
| **$F_x$:** | X-axis speed | **S (Xs,Ys):** | Current position (Start) |
| **$F_y$:** | Y-axis speed | **P (Xp,Yp)** | Pass position |
| **r:** | Radius | **E (Xe,Ye)** | Target position (End) |

$$F_x = F_v \sin\theta = F_v \frac{|Ye - Yo|}{r} \qquad F_y = F_v \cos\theta = F_v \frac{|Xe - Xo|}{r}$$

Example

Let channel 0 be the X-axis and channel 2 be the Y-axis.The position control mode is absolute value control.

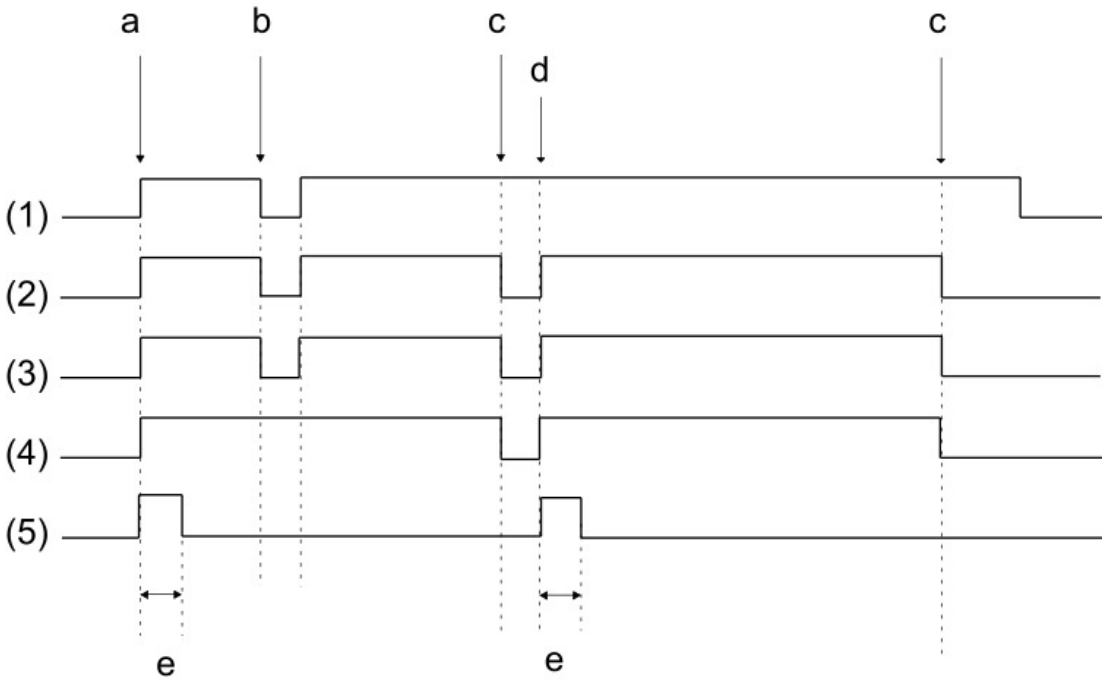
The current position is (θ=60°, Xs=5000, Ys=8660). The center position O (Xo=0, Yo=0) is used as a reference point. Pulses are output from the X-axis (channel 0) and the Y-axis (channel 2) at a speed of Fv=2000Hz until the target position (θ=-30°, Xe=8660, Ye=-5000) is reached.

## General programming information

- The execution condition for this instruction must be continually TRUE. When the execution condition is FALSE, pulse output stops.
- The high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- Executing the circular interpolation control instruction **F176_PulseOutput_Center** sets the circular interpolation control flag (**sys_bIsCircularInterpolationActive**) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed. To restart circular interpolation, perform a forced stop (stop pulse output) to set the circular interpolation control flag (**sys_bIsCircularInterpolationActive**) to FALSE.
- If "Continue" has been selected for the operation connection mode, use a special flag (**sys_bIsCircularInterpolationOverwritingPossible**) to permit overwriting of the target value. The flag is TRUE for one scan when the circular interpolation instruction is executed.
- The target value for each axis must be within the range of -8388608–8388607. When this instruction is used in combination with other pulse output instructions, e.g. **F171_PulseOutput_Trapezoidal**, the target value in these instructions must be within the same range.
- The accuracy of circular interpolation may degrade if the scan time is too long.
- Online editing during RUN mode is not available for this instruction.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- If you specify the same value for the current position and the target position, a circle drawing operation will result.
- As there is no interpolation function for the home return, the home return should be executed for each channel.
- When using in applications requiring precision, test runs with the actual machine are necessary.

- Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

### Flag setting during instruction execution




| | |
|---|---|
| (1) | Execution condition X0 |
| (2) | Pulse output control flag, channel 0 (**sys_bIsPulseChannel0Active**) |
| (3) | Pulse output control flag, channel 2 (**sys_bIsPulseChannel2Active**) |
| (4) | Circular interpolation control flag (**sys_bIsCircularInterpolationActive**) |
| (5) | Target value overwriting possible flag (**sys_bIsCircularInterpolationOverwritingPossible**) |
| **a** | Start |
| **b** | Execution condition FALSE |
| **c** | Target value reached |
| **d** | Start continue mode |
| **e** | 1 scan |

## Example

### POU header

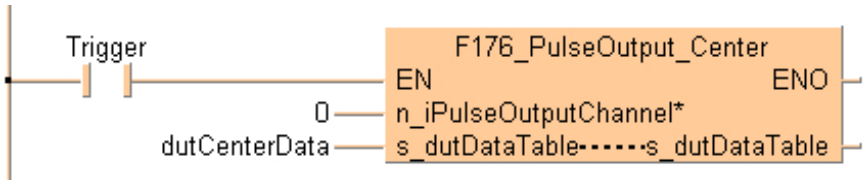
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | C... | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | Trigger | BOOL | FALSE | |
| 1 | VAR | dutCenterData | F176_PulseOutput_Center_DUT | dwControlCode := 16#01012, | |
| 2 | VAR | | | | |

dwControlCode := 16#01012,
diSpeed := 15000,
diTargetPos_X := 8000000,
diTargetPos_Y := 5000000,
diCenterPos_X := 8300000,
diCenterPos_Y := 5500000

Control code :
Digit 4: 0=Operation connection mode: stop
Digit 3: 1=Rotation direction: CCW (left)
Digit 2: 0=Fixed
Digit 1: 1=Absolute value control
Digit 0: 2=Pulse/direction (Forward FALSE)

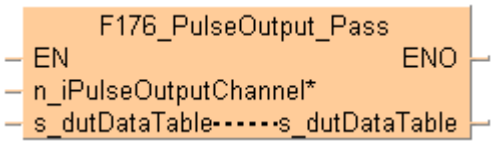
### LD body



# F176_PulseOutput_Pass

Circular interpolation (pass position)

Pulses are output from two channels in accordance with the parameters in the specified DUT, so that the path to the target position forms an arc.The radius of the circle is calculated by specifying the center position and the end position.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**n_iPulseOutputChannel*** (decimal constant)

Pulse output channel:0, 2

Input/output

**s_dutDataTable (F176_PulseOutput_Pass_DUT)**

Starting address of area containing the data table

## Remarks

Use the following predefined DUT: **F176_PulseOutput_Pass_DUT**

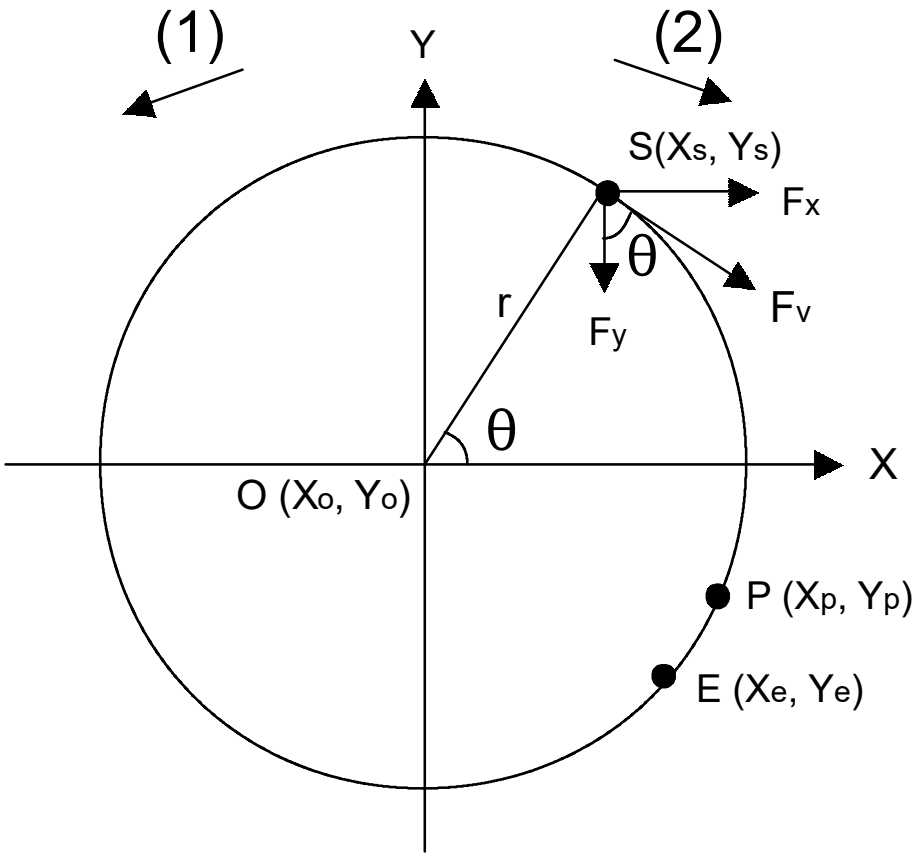
The following parameters can be specified in the DUT:

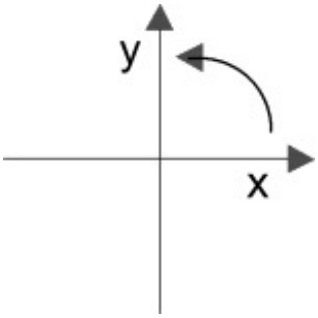
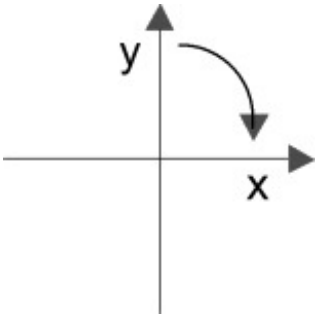
- Control code
- Composite speed
- X-axis target value
- Y-axis target value
- X-axis pass value
- Y-axis pass value

The following parameters for each axis are calculated upon execution of the instruction and stored in the operation result area of the DUT.

- Radius
- X-axis center value
- Y-axis center value

Pulse output characteristics




| | | | |
|---|---|---|---|
| (1) | Rotation direction: from channel 0 (x-axis) to channel 2 (y-axis) (for a movement in positive direction on both channels)   | (2) | Rotation direction: from channel 2 (y-axis) to channel 0 ( x-axis) (for a movement in positive direction on both channels)   |
| **$F_v$**: | Composite speed | **O (Xo,Yo):** | Center position |
| **$F_x$**: | X-axis speed | **S (Xs,Ys):** | Current position (start) |
| **$F_y$**: | Y-axis speed | **P (Xp,Yp)** | Pass position |
| **r:** | Radius | **E (Xe,Ye)** | Target position (End) |

$$F_x = F_v \sin\theta = F_v \frac{|Ye - Yo|}{r} \qquad F_y = F_v \cos\theta = F_v \frac{|Xe - Xo|}{r}$$

Example

Let channel 0 be the X-axis and channel 2 be the Y-axis.The position control mode is absolute value control.

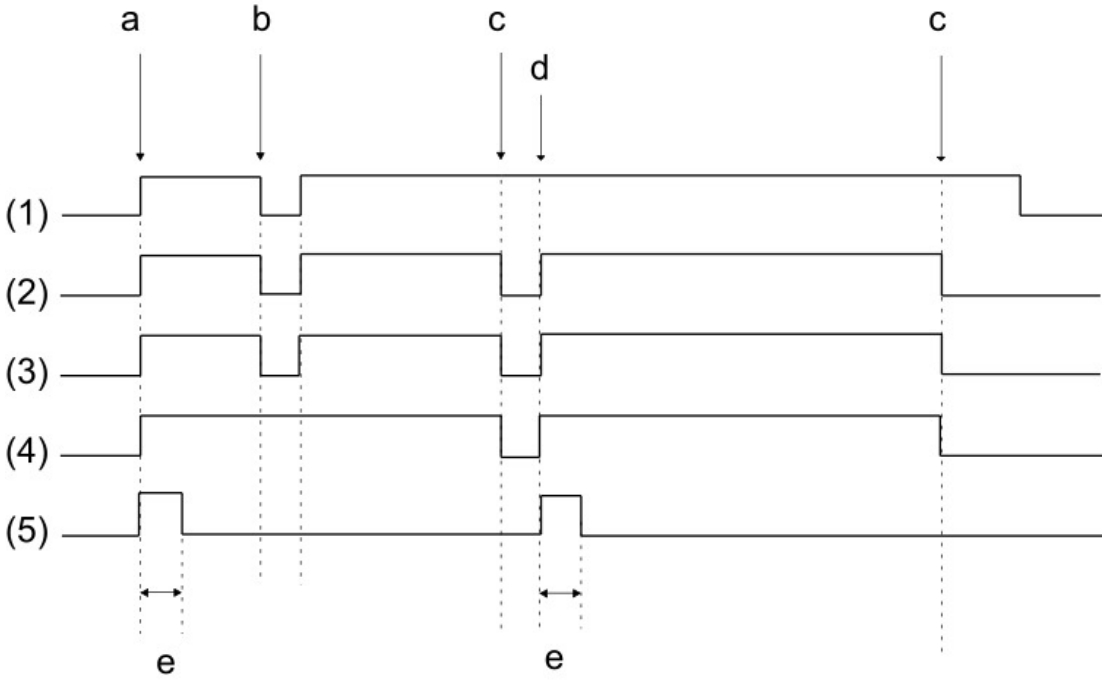
The current position is (θ=60°, Xs=5000, Ys=8660). The center position O (Xo=0, Yo=0) is used as a reference point. Pulses are output from the X-axis (channel 0) and the Y-axis (channel 2) at a speed of Fv=2000Hz until the target position (θ=-30°, Xe=8660, Ye=-5000) is reached.

## General programming information

- The execution condition for this instruction must be continually TRUE. When the execution condition is FALSE, pulse output stops.
- The high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) are assigned to the same special internal flag number (e.g. R903A). Therefore, when a high-speed counter instruction or a pulse output instruction is executed, both the high-speed counter control flag (e.g. **sys_bIsHscChannel0ControlActive**) and the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) for the channel used are TRUE. No other high-speed counter instruction or pulse output instruction can be executed as long as this flag is TRUE.
- Executing the circular interpolation control instruction **F176_PulseOutput_Center** sets the circular interpolation control flag (**sys_bIsCircularInterpolationActive**) to TRUE. The status of this flag is maintained until the target value is reached (even if the execution condition is no longer TRUE). During this time, other pulse output instructions cannot be executed. To restart circular interpolation, perform a forced stop (stop pulse output) to set the circular interpolation control flag (**sys_bIsCircularInterpolationActive**) to FALSE.
- If "Continue" has been selected for the operation connection mode, use a special flag (**sys_bIsCircularInterpolationOverwritingPossible**) to permit overwriting of the target value. The flag is TRUE for one scan when the circular interpolation instruction is executed.
- The target value for each axis must be within the range of -8388608–8388607. When this instruction is used in combination with other pulse output instructions, e.g. **F171_PulseOutput_Trapezoidal**, the target value in these instructions must be within the same range.
- The accuracy of circular interpolation may degrade if the scan time is too long.
- Online editing during RUN mode is not available for this instruction.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- As there is no interpolation function for the home return, the home return should be executed for each channel.
- When using in applications requiring precision, test runs with the actual machine are necessary.
- Set any high-speed counter allocated to a pulse output channel to "Unused" in the system registers.

- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

### Flag setting during instruction execution




| | |
|---|---|
| (1) | Execution condition X0 |
| (2) | Pulse output control flag, channel 0 (**sys_bIsPulseChannel0Active**) |
| (3) | Pulse output control flag, channel 2 (**sys_bIsPulseChannel2Active**) |
| (4) | Circular interpolation control flag (**sys_bIsCircularInterpolationActive**) |
| (5) | Target value overwriting possible flag (**sys_bIsCircularInterpolationOverwritingPossible**) |
| **a** | Start |
| **b** | Execution condition FALSE |
| **c** | Target value reached |
| **d** | Start continue mode |
| **e** | 1 scan |

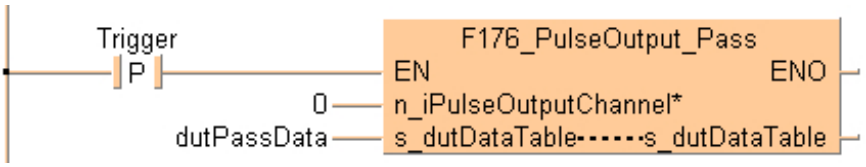
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | Trigger | BOOL | FALSE | |
| 1 | VAR | dutPassData | F176_PulseOutput_Pass_DUT | dwControlCode := 16#1000, | Control code:<br>Digit 4: 0=Operation connection mode: stop<br>Digit 3: 1=Rotation direction: CCW (left)<br>Digit 2: 0=Fixed<br>Digit 1: 0=Relative value control<br>Digit 0: 2=Pulse/direction (Forward FALSE) |
| 2 | VAR | | | dwControlCode := 16#1000,<br>diSpeed := 2000,<br>diTargetPos_X := 8660,<br>diTargetPos_Y := -5000,<br>diPassPos_X := 9396,<br>diPassPos_Y := -3420 | |

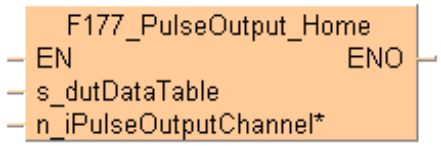
## LD body



# F177_PulseOutput_Home

Home return

This instruction performs a home return according to the parameters in the specified DUT.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**s_dutDataTable (F177_PulseOutput_Home_Type0_DUT)** or **(F177_PulseOutput_Home_Type1_DUT)**

Starting address of area containing the data table

**n_iPulseOutputChannel*** (decimal constant)

Pulse output channel:0–3

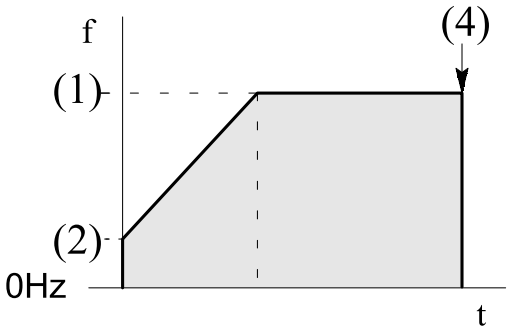
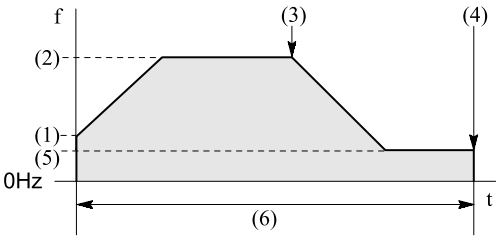
## Remarks

After a drive system has been switched on, there is a difference between the internal position value (elapsed value) and the mechanical position of the axis; this difference cannot be predetermined. The internal value must be synchronized with the actual position value of the axis. This is done by means of a home return, during which a position value is registered at a known reference point (home).During execution of a home return instruction, pulses are continuously output until the home input is enabled. The I/O allocation is determined by the channel used.To decelerate movement when near the home position, designate a near home input and set bit 4 of the special data register storing the pulse output control code (sys_wHscOrPulseControlCode) to TRUE and back to FALSE again.The deviation counter clear output can be set to TRUE when home return has been completed.

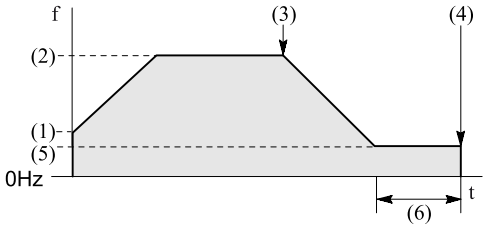
Select one of two different operation modes:

- Type 0:

  The home input is effective regardless of whether or not there is a near home input, whether deceleration is taking place, or whether deceleration has been completed.

| Without near home input: | With near home input: |
|---|---|
|  |  |
| (1) Initial speed<br>(2) Target speed<br>(3) -<br>(4) Home input: TRUE | (1) Initial speed<br>(2) Target speed<br>(3) Near home input: TRUE<br>(4) Home input: TRUE<br>(5) Creep speed<br>(6) Home input is effective at any time. |

- Type 1: The home input is effective only after deceleration (started by near home input) has been completed.




(1) Initial speed
(2) Target speed
(3) Near home input: TRUE
(4) Home input: TRUE
(5) Creep speed
(6) Home input is effective only after deceleration

Use the following predefined DUT: **F177_PulseOutput_Home_Type0_DUT** or **F177_PulseOutput_Home_Type1_DUT**

The following parameters can be specified in the DUT:

- Control code
- Initial speed
- Target speed
- Acceleration time
- Deceleration time
- Creep speed
- Deviation counter clear signal (output time)

Pulse output characteristics

- The pulse output frequency changes according to the specified acceleration time and the specified deceleration time.
- The difference between target and initial speed determines the slope of the ramps.
- Pulses are output using a duty of 25%.
- With the pulse output method "pulse/direction", pulses are output approx. 300μs after the direction signal has been output; the motor driver characteristics are simultaneously taken into consideration.

General programming information

- Set "Pulse output" for the desired channel in the system registers.
- Even when home input has occurred, executing this instruction causes pulse output to begin.
- If the near home input is enabled while acceleration is in progress, deceleration will start.
- The deviation counter clear signal is allocated to dedicated output numbers specific to each PLC type.
- If both the main program and the interrupt program contain code for the same channel, make sure both are not executed simultaneously.
- When a pulse output instruction is executed and pulses are being output, the pulse output control flag (e.g. **sys_bIsPulseChannel0Active**) of the corresponding channel is TRUE. No other pulse output instruction can be executed as long as this flag is TRUE.
- When programs are being edited in RUN mode, pulse output stops but resumes after the program changes have been downloaded.
- We strongly recommend that you incorporate a forced stop option in your positioning program.
- The status of the high-speed counter control flag or pulse output control flag may change while a scan is being carried out. For example, if the number of received bytes is read more than once different statuses may exist within one scan.

### Deviation counter clear outputs and home inputs for FP0R

- FP0R C16

| Channel no. | Deviation counter clear output | Home input |
|---|---|---|
| 0 | Y6 | X4 |
| 1 | Y7 | X5 |
| 2 | – | X6 |
| 3 | – | X7 |

Note

– Inputs X4–X7 can either be used as high-speed counter inputs or as home inputs.

– Y6 and Y7 can either be used as pulse outputs for channel 3 or as deviation counter clear outputs for channel 0 and 1.

- FP0R C32, T32, F32

| Channel no. | Deviation counter clear output | Home input |
|---|---|---|
| 0 | Y8 | X4 |
| 1 | Y9 | X5 |
| 2 | YA | X6 |
| 3 | YB | X7 |

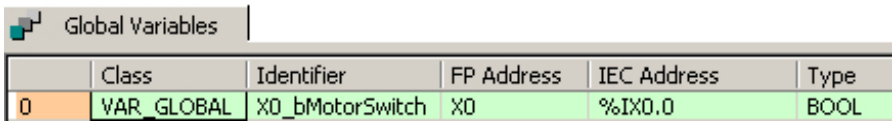
Note

Inputs X4–X7 can either be used as high-speed counter inputs or as home inputs.

## Example

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.

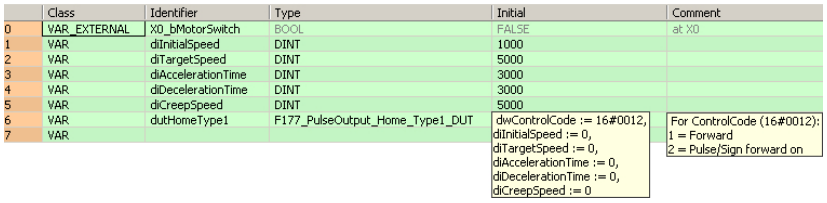



### DUT

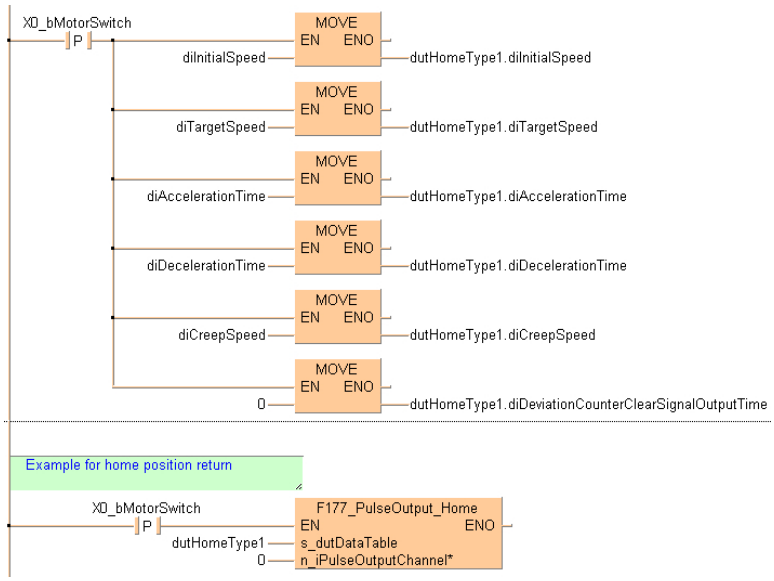
The DUT **F177_PulseOutput_Home_Type1_DUT** is predefined in the “FP library”.

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.



### LD body




## 26.3.5 Tool instructions for pulse output

| Pulse output instructions | | FPΣ | FP-X | FP0R | FPe, FP0 |
|---|---|---|---|---|---|
| Trapezoidal control | PulseOutput_Trapezoidal_FB | ● | ● | ● | ● |
| Home return | PulseOutput_Home_FB | ● | ● | ● | ● |
| JOG operation | PulseOutput_Jog_FB | ● | ● | ● | ● |
| JOG operation with target value | PulseOutput_Jog_TargetValue_FB | ● | ● | ● | |
| JOG operation and positioning | PulseOutput_Jog_Positioning0_FB | | | ● | |
| | PulseOutput_Jog_Positioning1_FB | | | ● | |
| Linear interpolation | PulseOutput_Linear_FB | ● | ● | ● | |
| Circular interpolation (center position) | PulseOutput_Center_FB | ● | | | |
| Circular interpolation (pass position) | PulseOutput_Pass_FB | ● | | | |

| Target value match instructions | | FPΣ | FP-X | FP0R | FPe, FP0 |
|---|---|---|---|---|---|
| Target value match ON | Pulse_TargetValueMatch_Set | | | ● | |
| Target value match OFF | Pulse_TargetValueMatch_Reset | | | ● | |

| Information instructions | FPΣ | FPX | FP0R | FPe, FP0 |
|---|---|---|---|---|
| Evaluating system register settings | | | | |
| PulseInfo_IsChannelEnabled | ● | ● | ● | ● |
| Evaluating and writing special internal flag and special data registers | | | | |
| PulseInfo_IsActive | ● | ● | ● | ● |
| PulseInfo_IsTargetValueMatchActive | | | ● | |
| PulseInfo_IsHomeInputTrue | ● | ● | ● | ● |
| PulseInfo_ReadElapsedValue | ● | ● | ● | ● |
| PulseInfo_ReadTargetValue | ● | ● | ● | ● |
| PulseInfo_GetCurrentSpeed | ● | ● | ● | ● |
| PulseInfo_ReadTargetValueMatchValue | | | ● | |
| PulseInfo_ReadAccelerationForbiddenAreaStartingPosition | | | ● | |
| PulseInfo_ReadCorrectedFinalSpeed | | | ● | |
| PulseInfo_ReadCorrectedInitialSpeed | | | ● | |
| Evaluating and writing the control code (DT90058) | | | | |
| PulseInfo_GetControlCode | ● | ● | ● | ● |
| PulseInfo_IsElapsedValueReset | ● | ● | ● | ● |
| PulseInfo_IsCountingDisabled | ● | ● | ● | ● |
| PulseInfo_IsPulseOutputStopped | ● | ● | ● | ● |

| Control instructions | | FPΣ | FPX | FP0R | FPe, FP0 |
|---|---|---|---|---|---|
| Evaluating and writing special internal flag and special data registers | | | | | |
| PulseControl_WriteElapsedValue | | ● | ● | ● | ● |
| Evaluating and writing the control code (DT90058) | | | | | |
| Set | Reset | | | | |
| | PulseControl_SetDefaults | ● | ● | ● | ● |
| PulseControl_ElapsedValueReset | PulseControl_ElapsedValueContinue | ● | ● | ● | ● |
| PulseControl_CountingDisable | PulseControl_CountingEnable | ● | ● | ● | ● |
| PulseControl_PulseOutputStop | PulseControl_PulseOutputContinue | ● | ● | ● | ● |
| PulseControl_TargetValueMatchClear | | | | ● | |
| PulseControl_NearHome | | ● | ● | ● | ● |
| PulseControl_DeceleratedStop | | | | ● | |
| PulseControl_JogPositionControl | | | | ● | |

### 26.3.5.1 Channel configuration DUT for all PLCs and all pulse output instructions

#### PulseOutput_Channel_Configuration_DUT

Use this DUT to make control code settings for all pulse output instructions. The universal DUT is a feature of the tool instructions for pulse output and applies to all PLCs.

**Elements of the DUT (identifiers):**

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

**bOutput_Pulse_ForwardTrue** (BOOL)

FP0R, FPΣ, FP-X: Pulse output method =Pulse/direction
Control is carried out using one pulse output to specify the speed and another to specify the direction of rotation with TRUE/FALSE signals. (Forward TRUE: In this mode, forward rotation is carried out when the rotation direction signal is TRUE.)
If neither **bOutput_Pulse_ForwardTrue** nor **bOutput_Pulse_ForwardFalse**: Control is carried out using two pulses: a positive or clockwise rotation pulse (CW) and a negative or counterclockwise rotation pulse (CCW pulse).
FP0, FP-e: Pulses only

**bOutput_Pulse_ForwardFalse** (BOOL)

FP0R, FPΣ, FP-X: Pulse output method = Pulse/direction
Control is carried out using one pulse output to specify the speed and another to specify the direction of rotation with TRUE/FALSE signals. (Forward FALSE: In this mode, forward rotation is carried out when the rotation direction signal is FALSE.)
If neither **bOutput_Pulse_ForwardTrue** nor **bOutput_Pulse_ForwardFalse**: Control is carried out using two pulses: a positive or clockwise rotation pulse (CW) and a negative or counterclockwise rotation pulse (CCW pulse).
FP0, FP-e: Pulses only

**bAccelerationSteps60** (BOOL)

FPΣ, FP-X: Number of acceleration/deceleration steps = 60 (else 30)

**bDutyRatio25** (BOOL)

FPΣ, FP-X: Duty ratio (for pulse duration and period).
The ratio between the pulse duration and the period of a rectangular waveform. For a pulse train in which the pulse duration is 1μs and the pulse period is 4μs, the duty ratio is 0.25 or 25%.

**bFrequencyRange_48Hz_100kHz** (BOOL)

FPΣ, FP-X: Frequency range for initial and target speed = 48Hz–100kHz (else 1.5Hz–9.8kHz)

**bFrequencyRange_191Hz_100kHz** (BOOL)

FPΣ, FP-X: Frequency range for initial and target speed = 191Hz–100kHz (else 1.5Hz–9.8kHz)

**bPulseWidth80µs** (BOOL)

FP0, FP-e: Pulse width = 80μs

**iDutyRatioIn10PercentSteps** (INT)

FP0, FP-e: Duty ratio (for pulse duration and period)
The ratio between the pulse duration and the period of a rectangular waveform. For a pulse train in which the pulse duration is 1μs and the pulse period is 4μs, the duty ratio is 0.25 or 25% = 10–90% (in steps of 10%)

**bEnableHomeOnlyAfterNearHomeDeceleration** (BOOL)

Operation mode

| | |
|---|---|
| FP0R: | Type 1<br>The home input is effective only after deceleration (started by near home input) has been completed.<br>(else type 0)<br>The home input is effective regardless of whether or not there is a near home input, whether deceleration is taking place, or whether deceleration has been completed. |
| FPΣ, FP-X: | Type 2<br>The home input is effective only after deceleration (started by near home input) has been completed.<br>(else type 1)<br>The home input is effective regardless of whether or not there is a near home input, whether deceleration is taking place, or whether deceleration has been completed. |

**iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms** (INT)

FP0R, FPΣ, FP-X: Deviation counter clear signal= 0–200 [x0.5ms]

Note

For **F171_PulseOutput_Home**, the instruction used internally, the valid range is 0–100 without conversion.

**bCalculationOnly** (BOOL)

FP0R (JOG operation, Trapezoidal control): Output operation = Calculation only (else Pulse output)

**bTrapezoidalMaximumTargetSpeed50kHz** (BOOL)

FP0R: Type 1 (The speed can be changed within the range of the maximum speed (50kHz).
Else type 0 (The speed can be changed within the range of the target speed specified first.)

**bExecuteInInterrupt** (BOOL)

FP0R (JOG operation, Trapezoidal control): Execute in interrupt program (else in main program)

**bJogWithNoCounting** (BOOL)

FP0, FP-e, FPΣ, FP-X: No counting (CW or CCW only), Mode with no target value

**bContinueAfterDone** (BOOL)

Operation connection mode: Use **sys_bIsCircularInterpolationOverwritingPossible**

## Related topics

Tool instructions for pulse output

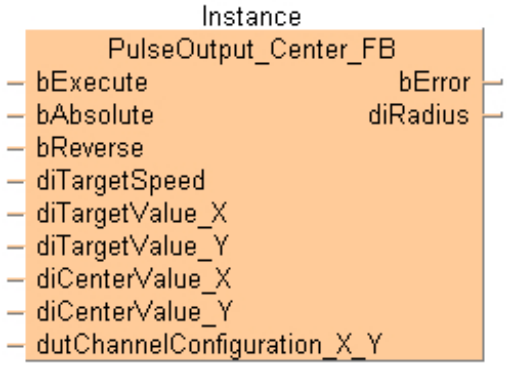
**F171_PulseOutput_Home**

### 26.3.5.2 Pulse output function blocks

# PulseOutput_Center_FB

Circular interpolation (center position)

Pulses are output from two channels in accordance with the parameters in the function block and in the specified DUT, so that the path to the target position forms an arc. The radius of the circle is calculated by specifying the center position and the end position. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**bExecute** (BOOL)

Activates the function block (with permanent trigger)

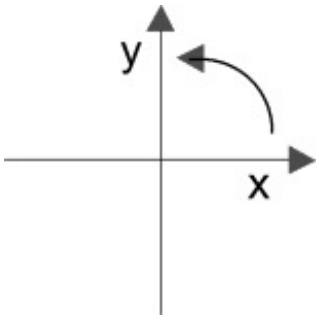
**bAbsolute** (BOOL)

Absolute value control = TRUE, Relative value control = FALSE

**bReverse** (BOOL)

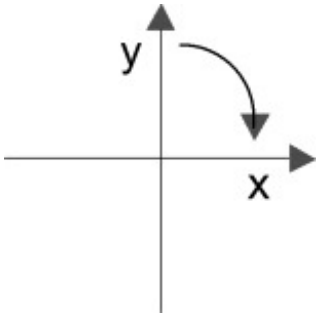
- TRUE=Rotation direction: reverse

  From channel 0 (x-axis) to channel 2 (y-axis) (for a movement in positive direction on both channels)




- FALSE=Rotation direction: forward

From channel 2 (y-axis) to channel 0 (x-axis) (for a movement in positive direction on both channels)




**diTargetSpeed** (DINT)

Target speed: Composite speed of both axes = 100–20000 (100Hz–20kHz)

**diTargetValue_X** (DINT)

X-axis target value [pulses]: -8388608–8388607

**diTargetValue_Y** (DINT)

Y-axis target value [pulses]: -8388608–8388607

**diCenterValue_X** (DINT)

X-axis center value [pulses]: -8388608–8388607

**diCenterValue_Y** (DINT)

Y-axis center value [pulses]: -8388608–8388607

**dutChannelConfiguration_X_Y**

Predefined system DUT for channel configuration:
**PulseOutput_Channel_Configuration_DUT**
Channel: 0, 2

Output

**bError** (BOOL)

TRUE if an applied input value is invalid. Execution of the function block stops.

**diRadius** (DINT)

Radius [pulses]

## Remarks

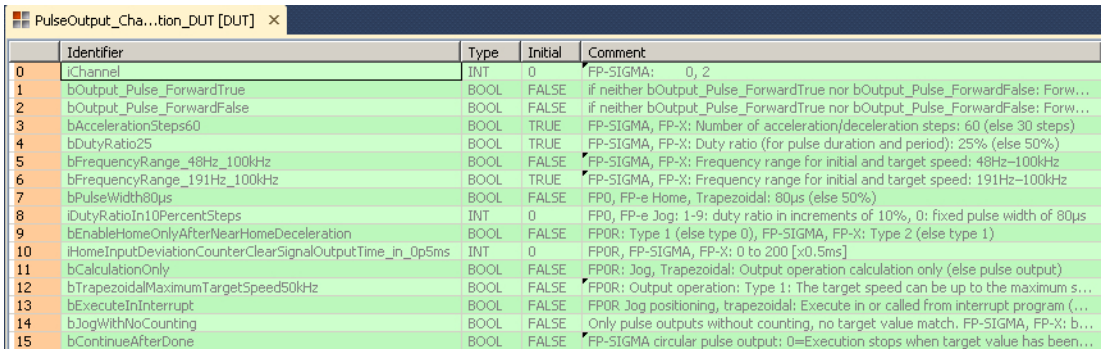
This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help:
**F176_PulseOutput_Center**

Use **PulseInfo_IsActive** to check if the control flag for the selected channel is FALSE.

*Example*

**DUT**

With a Data Unit Type (DUT) you can define a data unit type that is composed of other data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.



PulseOutput_Cha...tion_DUT [DUT]

| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | iChannel | INT | 0 | FP-SIGMA: 0, 2 |
| 1 | bOutput_Pulse_ForwardTrue | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 2 | bOutput_Pulse_ForwardFalse | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 3 | bAccelerationSteps60 | BOOL | TRUE | FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps) |
| 4 | bDutyRatio25 | BOOL | TRUE | FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%) |
| 5 | bFrequencyRange_48Hz_100kHz | BOOL | FALSE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz |
| 6 | bFrequencyRange_191Hz_100kHz | BOOL | TRUE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz |
| 7 | bPulseWidth80µs | BOOL | FALSE | FP0, FP-e Home, Trapezoidal: 80µs (else 50%) |
| 8 | iDutyRatioIn10PercentSteps | INT | 0 | FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs |
| 9 | bEnableHomeOnlyAfterNearHomeDeceleration | BOOL | FALSE | FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1) |
| 10 | iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms | INT | 0 | FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms] |
| 11 | bCalculationOnly | BOOL | FALSE | FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output) |
| 12 | bTrapezoidalMaximumTargetSpeed50kHz | BOOL | FALSE | FP0R: Output operation: Type 1: The target speed can be up to the maximum s... |
| 13 | bExecuteInInterrupt | BOOL | FALSE | FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (... |
| 14 | bJogWithNoCounting | BOOL | FALSE | Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b... |
| 15 | bContinueAfterDone | BOOL | FALSE | FP-SIGMA circular pulse output: 0=Execution stops when target value has been... |

**POU header**

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | PulseOutput_Center | PulseOutput_Center_FB | |
| 1 | VAR | bExecute | BOOL | FALSE |
| 2 | VAR | bAbsolute | BOOL | FALSE |
| 3 | VAR | bContinueAfterDone | BOOL | FALSE |
| 4 | VAR | bCounterclockwise | BOOL | FALSE |
| 5 | VAR | ChannelConfiguration_XY_DUT | PulseOutput_Channel_Configuration_DUT | |
| 6 | VAR | bError | BOOL | FALSE |
| 7 | VAR | diRadius | DINT | 0 |
| 8 | VAR | bConfigure | BOOL | FALSE |

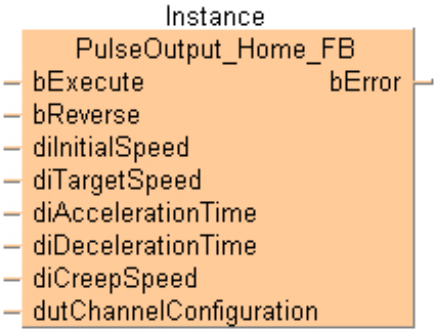
## LD body


bConfigure
MOVE
EN ENO
1
ChannelConfiguration_XY_DUT.iChannel
TRUE
ChannelConfiguration_XY_DUT.bOutput_Pulse_ForwardTrue
FALSE
ChannelConfiguration_XY_DUT.bOutput_Pulse_ForwardFalse
ChannelConfiguration_XY_DUT.bAccelerationSteps60
ChannelConfiguration_XY_DUT.bDutyRatio25
ChannelConfiguration_XY_DUT.bFrequencyRange_191Hz_100kHz
PulseOutput_Center
PulseOutput_Center_FB
bExecute
bAbsolute
bCounterclockwise
15000
7000
12000
500
900
ChannelConfiguration_XY_DUT
bReverse
diTargetSpeed
diTargetValue_X
diTargetValue_Y
diCenterValue_X
diCenterValue_Y
dutChannelConfiguration_X_Y
bError
diRadius

# PulseOutput_Home_FB

Home return

This instruction performs a home return according to the parameters in the function block and in the specified DUT. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**bExecute** (BOOL)

A rising edge activates the function block

**bReverse** (BOOL)

Movement direction: Forward = FALSE, Reverse = TRUE

**diInitialSpeed**, **diTargetSpeed** (DINT)

Initial speed/Target speed: Set this value according to the frequency range selected in **PulseOutput_Channel_Configuration_DUT**:
FPΣ, FP-X: 1 to 9800 (1.5Hz–9.8kHz)
48 to 100000 (48Hz–100kHz)
191 to 100000 (191–100kHz)
**F171_PulseOutput_Trapezoidal**: 1 to 50000 (1Hz–50kHz)
FP0, **F168_PulseOutput_Trapezoidal**: 40 to 5000 (40Hz–5kHz)

**diAccelerationTime** (DINT)

Acceleration/deceleration time (FPΣ, FP-X):
With 30 steps: 30ms–32760ms (specify in steps of 30)
With 60 steps: 60ms–32760ms (specify in steps of 60)
Acceleration/deceleration time (FP0, **F168_PulseOutput_Trapezoidal**): 30ms–32760ms
Acceleration time (**F171_PulseOutput_Trapezoidal**): 1ms–32760ms

**diDecelerationTime** (DINT)

Deceleration time (**F171_PulseOutput_Trapezoidal**): 1ms–32760ms

**diCreepSpeed** (DINT)

Creep speed (**F171_PulseOutput_Trapezoidal**): 1 to 50000 (1Hz–50kHz)

**dutChannelConfiguration**Predefined system DUT for channel configuration: **PulseOutput_Channel_Configuration_DUT**

Output

**bError** (BOOL)

TRUE if an applied input value is invalid. Execution of the function block stops.

## Remarks

This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help:

- FPΣ, FP-X: **F171_PulseOutput_Home**
- FP-e, FP0: **F168_PulseOutput_Home**

Use **PulseInfo_IsActive** to check if the control flag for the selected channel is FALSE.Use PulseInfo_IsHomeInputTrue to check if the home input is TRUE.

Note

Avoid malfunctions or an operation error:

- Make sure to set the system register to pulse output mode when using a home input.
- The home input may not be occupied by other instructions like pulse-catch input, interrupt input or high-speed counter.

*Example*

## DUT

With a Data Unit Type (DUT) you can define a data unit type that is composed of other data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

PulseOutput_Cha...tion_DUT [DUT]

| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | iChannel | INT | 0 | FP-SIGMA: 0, 2 |
| 1 | bOutput_Pulse_ForwardTrue | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 2 | bOutput_Pulse_ForwardFalse | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 3 | bAccelerationSteps60 | BOOL | TRUE | FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps) |
| 4 | bDutyRatio25 | BOOL | TRUE | FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%) |
| 5 | bFrequencyRange_48Hz_100kHz | BOOL | FALSE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz |
| 6 | bFrequencyRange_191Hz_100kHz | BOOL | TRUE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz |
| 7 | bPulseWidth80µs | BOOL | FALSE | FP0, FP-e Home, Trapezoidal: 80µs (else 50%) |
| 8 | iDutyRatioIn10PercentSteps | INT | 0 | FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs |
| 9 | bEnableHomeOnlyAfterNearHomeDeceleration | BOOL | FALSE | FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1) |
| 10 | iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms | INT | 0 | FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms] |
| 11 | bCalculationOnly | BOOL | FALSE | FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output) |
| 12 | bTrapezoidalMaximumTargetSpeed50kHz | BOOL | FALSE | FP0R: Output operation: Type 1: The target speed can be up to the maximum s... |
| 13 | bExecuteInInterrupt | BOOL | FALSE | FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (... |
| 14 | bJogWithNoCounting | BOOL | FALSE | Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b... |
| 15 | bContinueAfterDone | BOOL | FALSE | FP-SIGMA circular pulse output: 0=Execution stops when target value has been... |

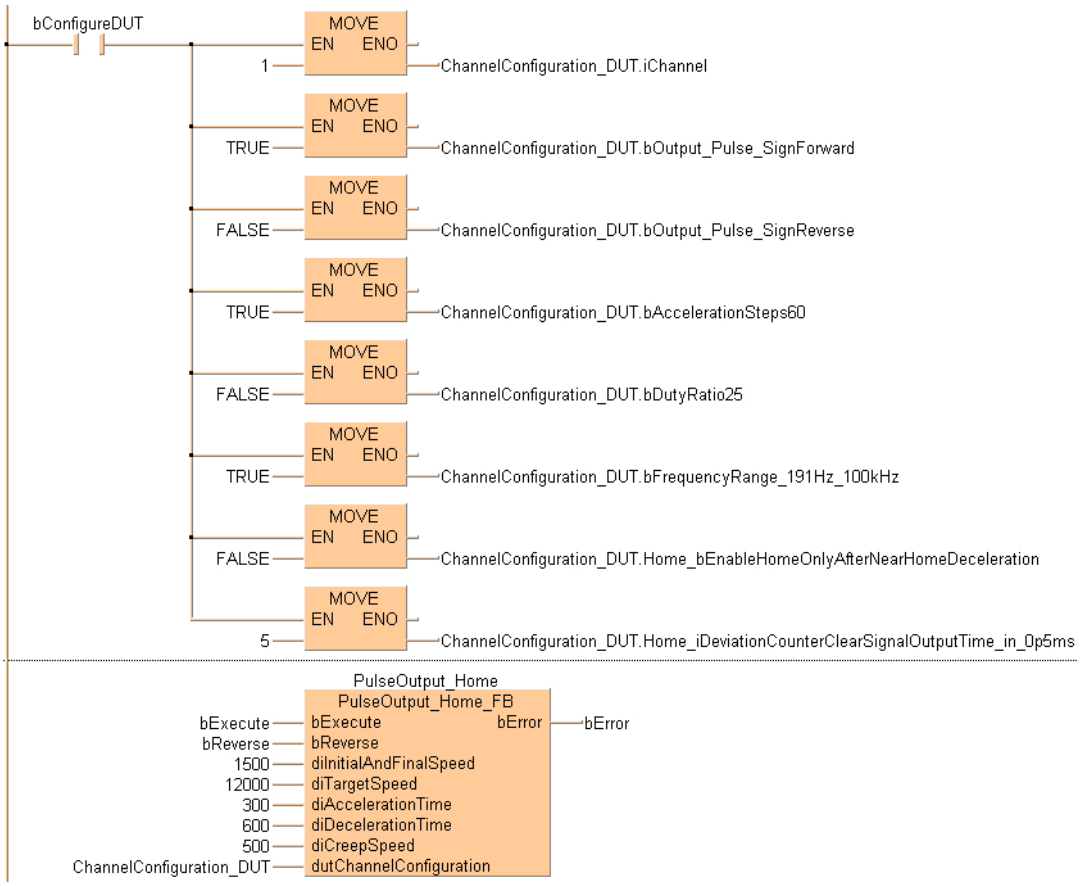
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | PulseOutput_Home | PulseOutput_Home_FB | |
| 1 | VAR | bExecute | BOOL | FALSE |
| 2 | VAR | bReverse | BOOL | FALSE |
| 3 | VAR | bError | BOOL | FALSE |
| 4 | VAR | ChannelConfiguration_DUT | PulseOutput_Channel_Configuration_DUT | |
| 5 | VAR | bConfigureDUT | BOOL | FALSE |

## POU body

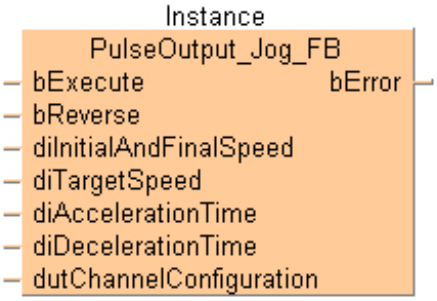
## LD body


bConfigureDUT
MOVE
EN ENO
1
ChannelConfiguration_DUT.iChannel
TRUE
ChannelConfiguration_DUT.bOutput_Pulse_SignForward
FALSE
ChannelConfiguration_DUT.bOutput_Pulse_SignReverse
TRUE
ChannelConfiguration_DUT.bAccelerationSteps60
FALSE
ChannelConfiguration_DUT.bDutyRatio25
TRUE
ChannelConfiguration_DUT.bFrequencyRange_191Hz_100kHz
FALSE
ChannelConfiguration_DUT.Home_bEnableHomeOnlyAfterNearHomeDeceleration
5
ChannelConfiguration_DUT.Home_iDeviationCounterClearSignalOutputTime_in_0p5ms
PulseOutput_Home
PulseOutput_Home_FB
bExecute
bReverse
1500
12000
300
600
500
ChannelConfiguration_DUT
bExecute
bReverse
diInitialAndFinalSpeed
diTargetSpeed
diAccelerationTime
diDecelerationTime
diCreepSpeed
dutChannelConfiguration
bError
bError

# PulseOutput_Jog_FB

JOG operation

This instruction is used for JOG operation.The specified number of pulses is output after the position control trigger input has turned to TRUE. A deceleration is performed before the target value is reached and pulse output stops.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**bExecute** (BOOL)

Execution condition can be:

- with edge trigger
- permanent, if change of speed is required.

**bReverse** (BOOL)

Movement direction: Forward = FALSE, Reverse = TRUE

**diInitialAndFinalSpeed** (DINT)

Initial and final speed (**F171_PulseOutput_Trapezoidal**): 1 to 50000 (1Hz–50kHz)

**diTargetSpeed** (DINT)

Target speed: Set this value according to the frequency range selected in **PulseOutput_Channel_Configuration_DUT**:
FPΣ, FP-X: 1 to 9800 (1.5Hz–9.8kHz)
48 to 100000 (48Hz–100kHz)
191 to 100000 (191–100kHz)
**F171_PulseOutput_Trapezoidal**: 1 to 50000 (1Hz–50kHz)
FP0, **F168_PulseOutput_Trapezoidal**: 40 to 5000 (40Hz–5kHz)

**diAccelerationTime** (DINT)

Acceleration time (**F171_PulseOutput_Trapezoidal**): 1ms–32760ms (up to the maximum speed)

**diDecelerationTime** (DINT)

Deceleration time (**F171_PulseOutput_Trapezoidal**): 1ms–32760ms (from the maximum speed)

**dutChannelConfiguration**Predefined system DUT for channel configuration: **PulseOutput_Channel_Configuration_DUT**

Output

**bError** (BOOL)

TRUE if an applied input value is invalid. Execution of the function block stops.

## Remarks

This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help:**F172_PulseOutput_Jog**.Use **PulseInfo_IsActive** to check if the control flag for the selected channel is FALSE.

*Example*

## DUT

With a Data Unit Type (DUT) you can define a data unit type that is composed of other data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

PulseOutput_Cha...tion_DUT [DUT]

| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | iChannel | INT | 0 | FP-SIGMA: 0, 2 |
| 1 | bOutput_Pulse_ForwardTrue | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 2 | bOutput_Pulse_ForwardFalse | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 3 | bAccelerationSteps60 | BOOL | TRUE | FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps) |
| 4 | bDutyRatio25 | BOOL | TRUE | FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%) |
| 5 | bFrequencyRange_48Hz_100kHz | BOOL | FALSE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz |
| 6 | bFrequencyRange_191Hz_100kHz | BOOL | TRUE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz |
| 7 | bPulseWidth80µs | BOOL | FALSE | FP0, FP-e Home, Trapezoidal: 80µs (else 50%) |
| 8 | iDutyRatioIn10PercentSteps | INT | 0 | FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs |
| 9 | bEnableHomeOnlyAfterNearHomeDeceleration | BOOL | FALSE | FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1) |
| 10 | iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms | INT | 0 | FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms] |
| 11 | bCalculationOnly | BOOL | FALSE | FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output) |
| 12 | bTrapezoidalMaximumTargetSpeed50kHz | BOOL | FALSE | FP0R: Output operation: Type 1: The target speed can be up to the maximum s... |
| 13 | bExecuteInInterrupt | BOOL | FALSE | FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (... |
| 14 | bJogWithNoCounting | BOOL | FALSE | Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b... |
| 15 | bContinueAfterDone | BOOL | FALSE | FP-SIGMA circular pulse output: 0=Execution stops when target value has been... |

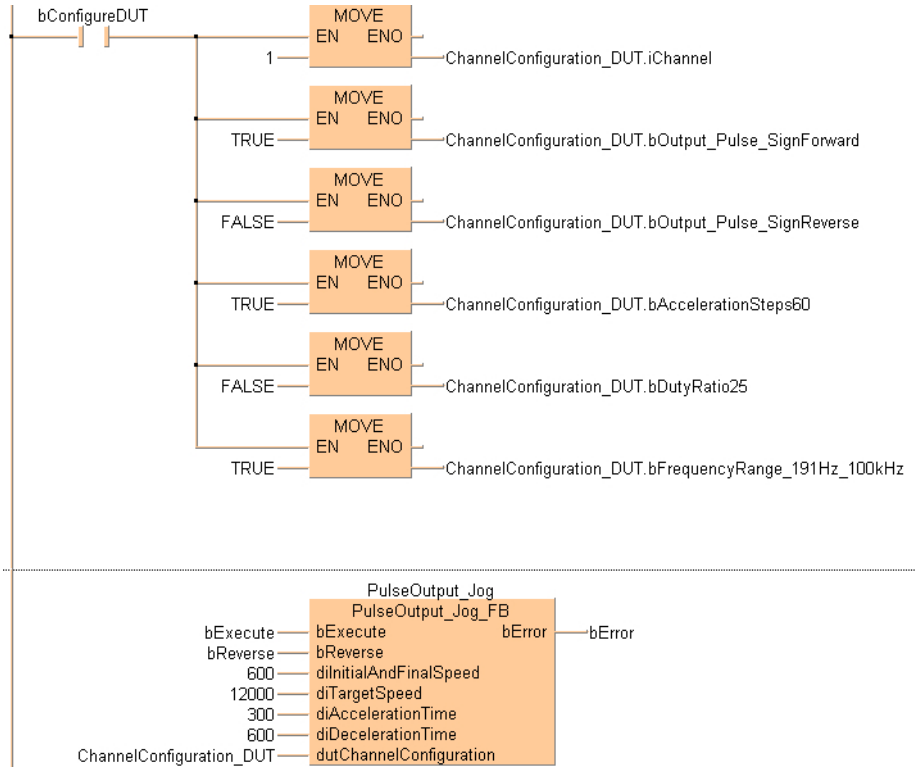
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | PulseOutput_Jog | PulseOutput_Jog_FB | |
| 1 | VAR | bExecute | BOOL | FALSE |
| 2 | VAR | bReverse | BOOL | FALSE |
| 3 | VAR | ChannelConfiguration_DUT | PulseOutput_Channel_Configuration_DUT | |
| 4 | VAR | bError | BOOL | FALSE |
| 5 | VAR | bConfigureDUT | BOOL | FALSE |

## POU body

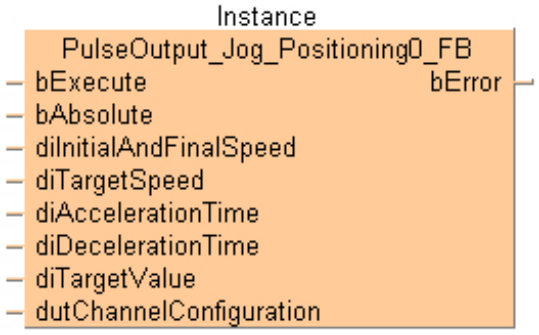
## LD body



# PulseOutput_Jog_Positioning0_FB

JOG operation and positioning

This instruction is used for JOG operation.The specified number of pulses is output after the position control trigger input has turned to TRUE. A deceleration is performed before the target value is reached and pulse output stops.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.The speed can be changed within the range of the specified target speed.




## Parameters

Input

**bExecute** (BOOL)

Execution condition can be:

- with edge trigger
- permanent, if change of speed is required.

**bAbsolute**BOOL:=FALSE

Only relative value control is supported; must always be FALSE, otherwise an error is output.

**diInitialAndFinalSpeed** (DINT)

Initial and final speed (**F171_PulseOutput_Trapezoidal**): 1 to 50000 (1Hz–50kHz)

**diTargetSpeed** (DINT)

Target speed: Set this value according to the frequency range selected in **PulseOutput_Channel_Configuration_DUT**:
FPΣ, FP-X: 1 to 9800 (1.5Hz–9.8kHz)
48 to 100000 (48Hz–100kHz)
191 to 100000 (191–100kHz)
**F171_PulseOutput_Trapezoidal**: 1 to 50000 (1Hz–50kHz)
FP0, **F168_PulseOutput_Trapezoidal**: 40 to 5000 (40Hz–5kHz)

**diAccelerationTime** (DINT)

Acceleration time (**F171_PulseOutput_Trapezoidal**): 1ms–32760ms (up to the maximum speed)

**diDecelerationTime** (DINT)

Deceleration time (**F171_PulseOutput_Trapezoidal**): 1ms–32760ms (from the maximum speed)

**diTargetValue** (DINT)

Target value[pulses]: -2147483648–2147483647

**dutChannelConfiguration**Predefined system DUT for channel configuration: **PulseOutput_Channel_Configuration_DUT**

Output

**bError** (BOOL)

TRUE if an applied input value is invalid. Execution of the function block stops.
TRUE if the applied channel is not enabled in the system registers or if **bAbsolute** is TRUE

## Remarks

This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help:**F171_PulseOutput_Jog_Positioning**. Use **PulseInfo_IsActive** to check if the control flag for the selected channel is FALSE.Use PulseControl_PulseOutputStop to stop pulse output on a specified channel. Use PulseControl_DeceleratedStop to perform a decelerated stop.

*Example*

## DUT

With a Data Unit Type (DUT) you can define a data unit type that is composed of other data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

PulseOutput_Cha...tion_DUT [DUT]

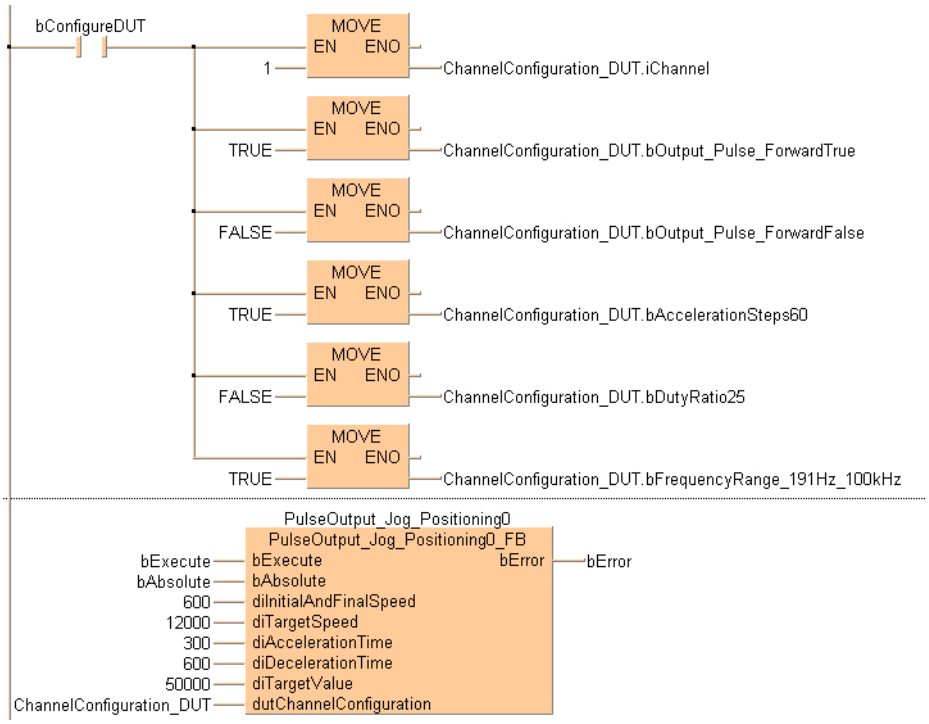
| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | iChannel | INT | 0 | FP-SIGMA: 0, 2 |
| 1 | bOutput_Pulse_ForwardTrue | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 2 | bOutput_Pulse_ForwardFalse | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 3 | bAccelerationSteps60 | BOOL | TRUE | FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps) |
| 4 | bDutyRatio25 | BOOL | TRUE | FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%) |
| 5 | bFrequencyRange_48Hz_100kHz | BOOL | FALSE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz |
| 6 | bFrequencyRange_191Hz_100kHz | BOOL | TRUE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz |
| 7 | bPulseWidth80µs | BOOL | FALSE | FP0, FP-e Home, Trapezoidal: 80µs (else 50%) |
| 8 | iDutyRatioIn10PercentSteps | INT | 0 | FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs |
| 9 | bEnableHomeOnlyAfterNearHomeDeceleration | BOOL | FALSE | FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1) |
| 10 | iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms | INT | 0 | FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms] |
| 11 | bCalculationOnly | BOOL | FALSE | FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output) |
| 12 | bTrapezoidalMaximumTargetSpeed50kHz | BOOL | FALSE | FP0R: Output operation: Type 1: The target speed can be up to the maximum s... |
| 13 | bExecuteInInterrupt | BOOL | FALSE | FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (... |
| 14 | bJogWithNoCounting | BOOL | FALSE | Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b... |
| 15 | bContinueAfterDone | BOOL | FALSE | FP-SIGMA circular pulse output: 0=Execution stops when target value has been... |

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | PulseOutput_Jog_Positioning0 | PulseOutput_Jog_Positioning0_FB | |
| 1 | VAR | bExecute | BOOL | FALSE |
| 2 | VAR | ChannelConfiguration_DUT | PulseOutput_Channel_Configuration_DUT | |
| 3 | VAR | bError | BOOL | FALSE |
| 4 | VAR | bConfigureDUT | BOOL | FALSE |
| 5 | VAR | bAbsolute | BOOL | FALSE |

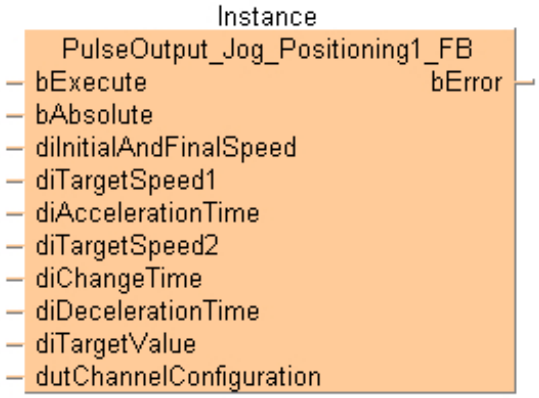
## LD body



# PulseOutput_Jog_Positioning1_FB

JOG operation and positioning

This instruction is used for JOG operation.The specified number of pulses is output after the position control trigger input has turned to TRUE. A deceleration is performed before the target value is reached and pulse output stops.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.The target speed can be changed once when the position control trigger input turns to TRUE.




## Parameters

Input

**bExecute** (BOOL)

Execution condition can be:

- with edge trigger
- permanent, if change of speed is required.

**bAbsolute**BOOL:=FALSE

Only relative value control is supported; must always be FALSE, otherwise an error is output.

**diInitialAndFinalSpeed** (DINT)

Initial and final speed = 1–50000 (1Hz–50kHz)

**diTargetSpeed1** (DINT)

Target speed = 1–50000 (1Hz–50kHz)

**diAccelerationTime** (DINT)

Acceleration time= 1ms–32760ms

**diTargetSpeed2** (DINT)

Target speed = 1–50000 (1Hz–50kHz)

**diChangeTime** (DINT)

Change time = 1–32760ms

**diDecelerationTime** (DINT)

Deceleration time = 1–32760ms

**diTargetValue** (DINT)

Target value[pulses]: -2147483648–2147483647

Output

**bError** (BOOL)

TRUE if an applied input value is invalid. Execution of the function block stops.
TRUE if the applied channel is not enabled in the system registers or if **bAbsolute** is TRUE

## Remarks

This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help:**F171_PulseOutput_Jog_Positioning**

Use **PulseInfo_IsActive** to check if the control flag for the selected channel is FALSE.Use PulseControl_PulseOutputStop to stop pulse output on a specified channel. Use PulseControl_DeceleratedStop to perform a decelerated stop.

*Example*

## DUT

With a Data Unit Type (DUT) you can define a data unit type that is composed of other data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

PulseOutput_Cha...tion_DUT [DUT]

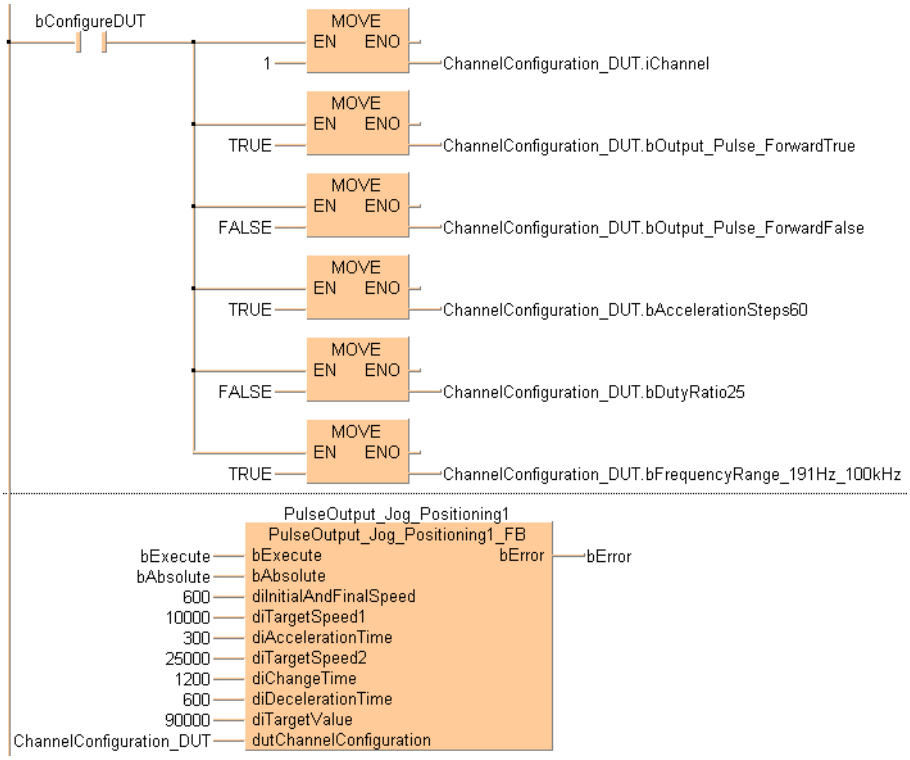
| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | iChannel | INT | 0 | FP-SIGMA: 0, 2 |
| 1 | bOutput_Pulse_ForwardTrue | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 2 | bOutput_Pulse_ForwardFalse | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 3 | bAccelerationSteps60 | BOOL | TRUE | FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps) |
| 4 | bDutyRatio25 | BOOL | TRUE | FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%) |
| 5 | bFrequencyRange_48Hz_100kHz | BOOL | FALSE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz |
| 6 | bFrequencyRange_191Hz_100kHz | BOOL | TRUE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz |
| 7 | bPulseWidth80µs | BOOL | FALSE | FP0, FP-e Home, Trapezoidal: 80µs (else 50%) |
| 8 | iDutyRatioIn10PercentSteps | INT | 0 | FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs |
| 9 | bEnableHomeOnlyAfterNearHomeDeceleration | BOOL | FALSE | FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1) |
| 10 | iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms | INT | 0 | FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms] |
| 11 | bCalculationOnly | BOOL | FALSE | FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output) |
| 12 | bTrapezoidalMaximumTargetSpeed50kHz | BOOL | FALSE | FP0R: Output operation: Type 1: The target speed can be up to the maximum s... |
| 13 | bExecuteInInterrupt | BOOL | FALSE | FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (... |
| 14 | bJogWithNoCounting | BOOL | FALSE | Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b... |
| 15 | bContinueAfterDone | BOOL | FALSE | FP-SIGMA circular pulse output: 0=Execution stops when target value has been... |

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | PulseOutput_Jog_Positioning1 | PulseOutput_Jog_Positioning1_FB | |
| 1 | VAR | bExecute | BOOL | FALSE |
| 2 | VAR | ChannelConfiguration_DUT | PulseOutput_Channel_Configuration_DUT | |
| 3 | VAR | bError | BOOL | FALSE |
| 4 | VAR | bConfigureDUT | BOOL | FALSE |
| 5 | VAR | bAbsolute | BOOL | FALSE |

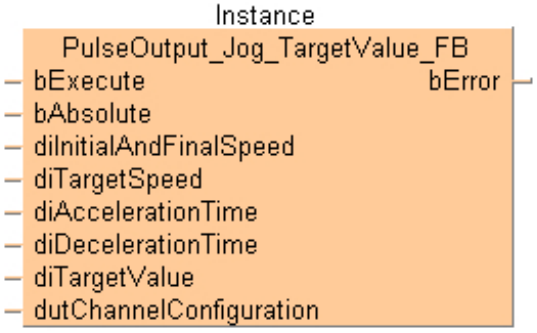
## LD body



# PulseOutput_Jog_TargetValue_FB

JOG operation with target value

This instruction is used for JOG operation.The specified number of pulses is output after the position control trigger input has turned to TRUE. A deceleration is performed before the target value is reached and pulse output stops.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE. Pulse output stops when the target value is reached.




## Parameters

Input

**bExecute** (BOOL)

Execution condition can be:

- with edge trigger
- permanent, if change of speed is required.

**bAbsolute** (BOOL)

**F171_PulseOutput_Trapezoidal**: Absolute value control = TRUE, Relative value control = FALSE

**diInitialAndFinalSpeed** (BOOL)

**F171_PulseOutput_Trapezoidal**: Initial and final speed = 1–50000 (1Hz–50kHz)

**diTargetSpeed** (DINT)

Target speed: Set this value according to the frequency range selected in **PulseOutput_Channel_Configuration_DUT**:
FPΣ, FP-X: 1–9800 (1.5Hz–9.8kHz)
48–100000 (48Hz–100kHz)
191–100000 (191–100kHz)
**F171_PulseOutput_Trapezoidal**: 1–50000 (1Hz–50kHz)

FP0, **F168_PulseOutput_Trapezoidal**: 40–5000 (40Hz–5kHz)

**diAccelerationTime** (DINT)

Acceleration time (**F171_PulseOutput_Trapezoidal**): 1ms–32760ms (up to the maximum speed)

**diDecelerationTime** (DINT)

Deceleration time (**F171_PulseOutput_Trapezoidal**): 1ms–32760ms (from the maximum speed)

**diTargetValue** (DINT)

Target value[pulses]: -2147483648–2147483647

**dutChannelConfiguration**Predefined system DUT for channel configuration: **PulseOutput_Channel_Configuration_DUT**

Output

**bError** (BOOL)

TRUE if an applied input value is invalid. Execution of the function block stops.
Additional error condition for FPΣ, FP-X:
TRUE if the applied channel is not enabled in the system registers or if **bAbsolute** is TRUE

## Remarks

This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help:**F172_PulseOutput_Jog**. Use **PulseInfo_IsActive** to check if the control flag for the selected channel is FALSE.

*Example*

## DUT

With a Data Unit Type (DUT) you can define a data unit type that is composed of other data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

PulseOutput_Cha...tion_DUT [DUT]

| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | iChannel | INT | 0 | FP-SIGMA: 0, 2 |
| 1 | bOutput_Pulse_ForwardTrue | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 2 | bOutput_Pulse_ForwardFalse | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 3 | bAccelerationSteps60 | BOOL | TRUE | FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps) |
| 4 | bDutyRatio25 | BOOL | TRUE | FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%) |
| 5 | bFrequencyRange_48Hz_100kHz | BOOL | FALSE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz |
| 6 | bFrequencyRange_191Hz_100kHz | BOOL | TRUE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz |
| 7 | bPulseWidth80µs | BOOL | FALSE | FP0, FP-e Home, Trapezoidal: 80µs (else 50%) |
| 8 | iDutyRatioIn10PercentSteps | INT | 0 | FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs |
| 9 | bEnableHomeOnlyAfterNearHomeDeceleration | BOOL | FALSE | FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1) |
| 10 | iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms | INT | 0 | FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms] |
| 11 | bCalculationOnly | BOOL | FALSE | FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output) |
| 12 | bTrapezoidalMaximumTargetSpeed50kHz | BOOL | FALSE | FP0R: Output operation: Type 1: The target speed can be up to the maximum s... |
| 13 | bExecuteInInterrupt | BOOL | FALSE | FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (... |
| 14 | bJogWithNoCounting | BOOL | FALSE | Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b... |
| 15 | bContinueAfterDone | BOOL | FALSE | FP-SIGMA circular pulse output: 0=Execution stops when target value has been... |

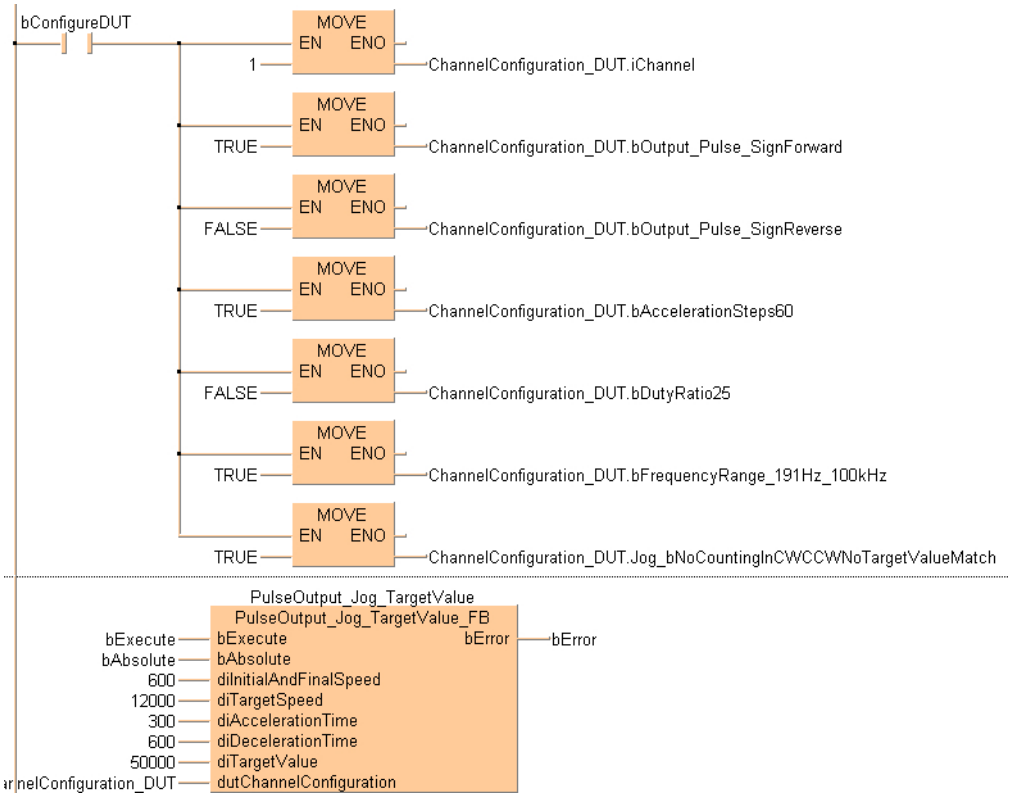
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | PulseOutput_Jog_TargetValue | PulseOutput_Jog_TargetValue_FB | |
| 1 | VAR | bExecute | BOOL | FALSE |
| 2 | VAR | bAbsolute | BOOL | FALSE |
| 3 | VAR | ChannelConfiguration_DUT | PulseOutput_Channel_Configuration_DUT | |
| 4 | VAR | bError | BOOL | FALSE |
| 5 | VAR | bConfigureDUT | BOOL | FALSE |

## POU body

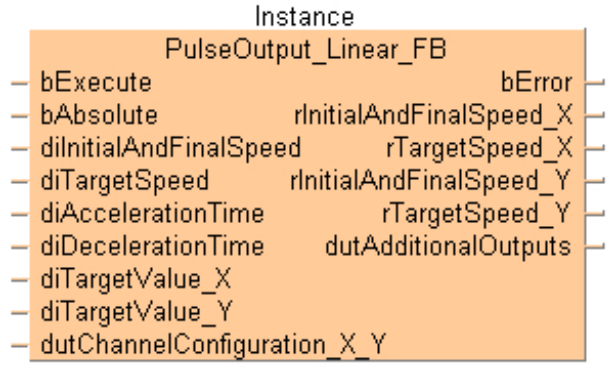
## LD body



# PulseOutput_Linear_FB

Linear interpolation

Pulses are output from two channels in accordance with the parameters in the function block and in the specified DUT, so that the path to the target position forms a straight line. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**bExecute** (BOOL)

Execution condition can be:

- with edge trigger
- permanent, if change of speed is required.

**bAbsolute** (BOOL)

Absolute value control = TRUE, Relative value control = FALSE

**diInitialAndFinalSpeed** (DINT)

Initial and final speed: Composite speed = 1–50000 (1Hz–50kHz)

**diTargetSpeed** (DINT)

Target speed: Composite speed = 1–50000 (1Hz–50kHz)

**diAccelerationTime** (DINT)

Acceleration/deceleration time (FPΣ, FP-X): 0ms–32767ms
Acceleration time (**F171_PulseOutput_Trapezoidal**): 0ms–32767ms

**diDecelerationTime** (DINT)

Deceleration time (**F171_PulseOutput_Trapezoidal**): 0ms–32767ms

**diTargetValue_X** (DINT)

X-axis target value[pulses]-8388608–8388607

**diTargetValue_Y** (DINT)

Y-axis target value[pulses]-8388608–8388607

**dutChannelConfiguration_X_Y**Predefined system DUT for channel configuration: **PulseOutput_Channel_Configuration_DUT**For interpolation, channel 0 and 1 or channel 2 and 3 are used as pairs. You may only specify 0 or 2 (for C14T: 0 only).

Output

**bError** (BOOL)

TRUE if an applied input value is invalid. Execution of the function block stops.
Is set only if global constant MC_PulseOutput_Library_Basic_bCheckInputs is set to TRUE.

**riInitialAndFinalSpeed_X** (REAL)

X-axis initial and final speed[Hz]

**riTargetSpeed_X** (REAL)

X-axis target speed[Hz]

**riInitialAndFinalSpeed_Y** (REAL)

Y-axis initial and final speed[Hz]

**riTargetSpeed_Y** (REAL)

Y-axis target speed[Hz]

**dutAdditionalOutputs**FPΣ, FP-X: PulseOutput_Linear_AdditionalOutputs_DUT

## Remarks

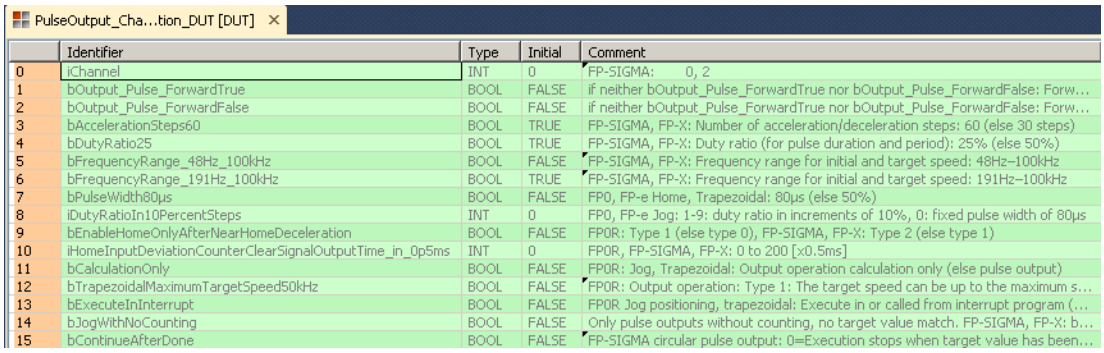
This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help:**F175_PulseOutput_Linear**

Use **PulseInfo_IsActive** to check if the control flag for the selected channel is FALSE.

*Example*

## DUT

With a Data Unit Type (DUT) you can define a data unit type that is composed of other data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.



PulseOutput_Cha...tion_DUT [DUT]

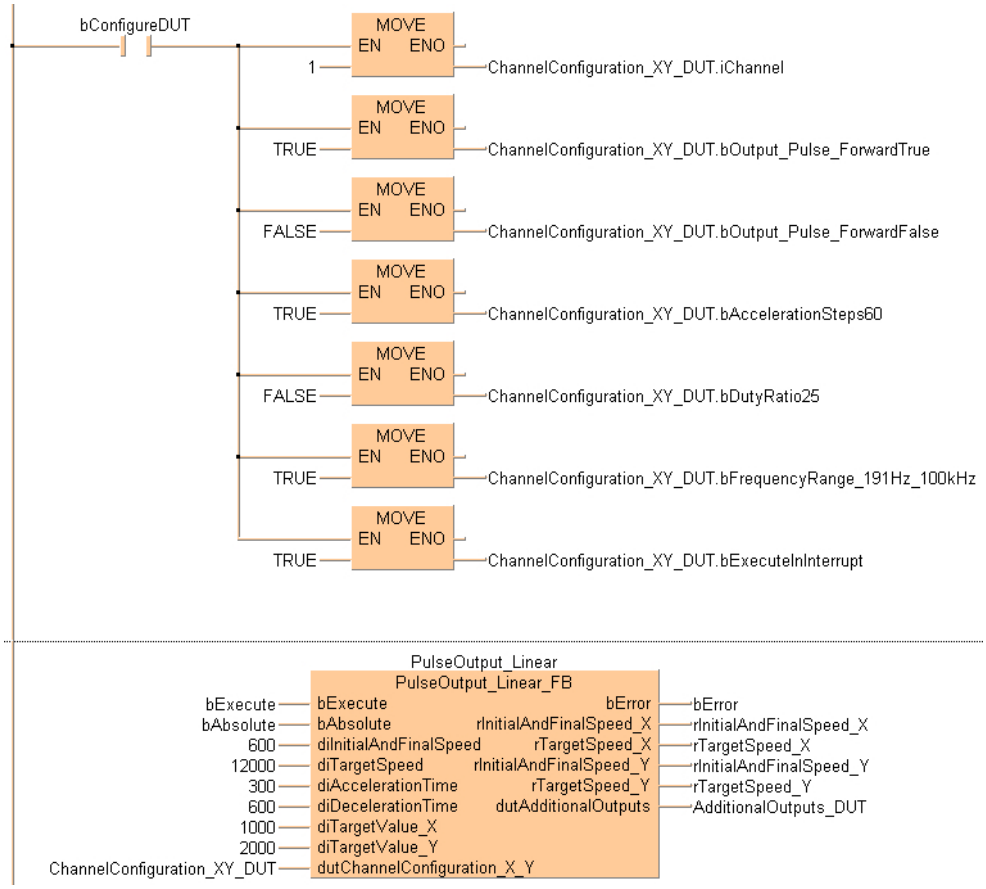
| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | iChannel | INT | 0 | FP-SIGMA: 0, 2 |
| 1 | bOutput_Pulse_ForwardTrue | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 2 | bOutput_Pulse_ForwardFalse | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 3 | bAccelerationSteps60 | BOOL | TRUE | FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps) |
| 4 | bDutyRatio25 | BOOL | TRUE | FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%) |
| 5 | bFrequencyRange_48Hz_100kHz | BOOL | FALSE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz |
| 6 | bFrequencyRange_191Hz_100kHz | BOOL | TRUE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz |
| 7 | bPulseWidth80µs | BOOL | FALSE | FP0, FP-e Home, Trapezoidal: 80µs (else 50%) |
| 8 | iDutyRatioIn10PercentSteps | INT | 0 | FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs |
| 9 | bEnableHomeOnlyAfterNearHomeDeceleration | BOOL | FALSE | FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1) |
| 10 | iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms | INT | 0 | FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms] |
| 11 | bCalculationOnly | BOOL | FALSE | FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output) |
| 12 | bTrapezoidalMaximumTargetSpeed50kHz | BOOL | FALSE | FP0R: Output operation: Type 1: The target speed can be up to the maximum s... |
| 13 | bExecuteInInterrupt | BOOL | FALSE | FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (... |
| 14 | bJogWithNoCounting | BOOL | FALSE | Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b... |
| 15 | bContinueAfterDone | BOOL | FALSE | FP-SIGMA circular pulse output: 0=Execution stops when target value has been... |

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | PulseOutput_Linear | PulseOutput_Linear_FB | |
| 1 | VAR | bExecute | BOOL | FALSE |
| 2 | VAR | bAbsolute | BOOL | FALSE |
| 3 | VAR | ChannelConfiguration_XY_D... | PulseOutput_Channel_Configuration_DUT | |
| 4 | VAR | bError | BOOL | FALSE |
| 5 | VAR | rInitialAndFinalSpeed_X | REAL | 0 |
| 6 | VAR | rTargetSpeed_X | REAL | 0 |
| 7 | VAR | rInitialAndFinalSpeed_Y | REAL | 0 |
| 8 | VAR | rTargetSpeed_Y | REAL | 0 |
| 9 | VAR | AdditionalOutputs_DUT | PulseOutput_Linear_AdditionalOutputs_DUT | |
| 10 | VAR | bConfigureDUT | BOOL | FALSE |

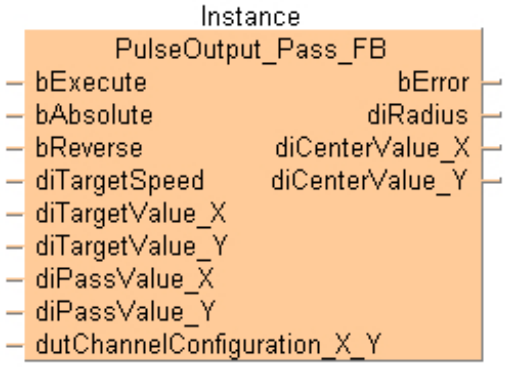
## LD body


bConfigureDUT
MOVE
EN ENO
1
ChannelConfiguration_XY_DUT.iChannel
TRUE
ChannelConfiguration_XY_DUT.bOutput_Pulse_ForwardTrue
FALSE
ChannelConfiguration_XY_DUT.bOutput_Pulse_ForwardFalse
ChannelConfiguration_XY_DUT.bAccelerationSteps60
ChannelConfiguration_XY_DUT.bDutyRatio25
ChannelConfiguration_XY_DUT.bFrequencyRange_191Hz_100kHz
ChannelConfiguration_XY_DUT.bExecuteInInterrupt
PulseOutput_Linear
PulseOutput_Linear_FB
bExecute
bAbsolute
600
12000
300
600
1000
2000
ChannelConfiguration_XY_DUT
diInitialAndFinalSpeed
diTargetSpeed
diAccelerationTime
diDecelerationTime
diTargetValue_X
diTargetValue_Y
dutChannelConfiguration_X_Y
bError
rInitialAndFinalSpeed_X
rTargetSpeed_X
rInitialAndFinalSpeed_Y
rTargetSpeed_Y
dutAdditionalOutputs
AdditionalOutputs_DUT

# PulseOutput_Pass_FB

Circular interpolation (pass position)

Pulses are output from two channels in accordance with the parameters in the function block and in the specified DUT, so that the path to the target position forms an arc. The radius of the circle is calculated by specifying the center position and the end position. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**bExecute** (BOOL)

Activates the function block

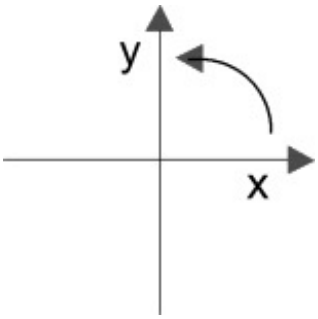
**bAbsolute** (BOOL)

Absolute value control = TRUE, Relative value control = FALSE

**bReverse** (BOOL)

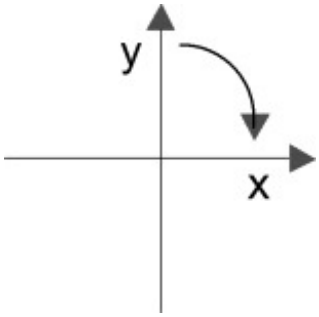
- TRUE=Rotation direction: reverse

  From channel 0 (x-axis) to channel 2 (y-axis) (for a movement in positive direction on both channels)




- FALSE=Rotation direction: forward

From channel 2 (y-axis) to channel 0 (x-axis) (for a movement in positive direction on both channels)




**diTargetSpeed** (DINT)

Target speed: Composite speed of both axes = 100–20000 (100Hz–20kHz)

**diTargetValue_X** (DINT)

Target value [pulses]: -8388608–8388607

**diTargetValue_Y** (DINT)

Target value [pulses]: -8388608–8388607

**diPassValue_X** (DINT)

Target value [pulses]: -8388608–8388607

**diPassValue_Y** (DINT)

Target value [pulses]: -8388608–8388607

**dutChannelConfiguration_X_Y**

Predefined system DUT for channel configuration:
**PulseOutput_Channel_Configuration_DUT**
Channel: 0, 2

Output

**bError** (BOOL)

TRUE if an applied input value is invalid. Execution of the function block stops.

**diRadius** (DINT)

Radius [pulses]

**diCenterValue_X** (DINT)

X-axis center value [pulses] = -8388608–8388607

**diCenterValue_Y** (DINT)

Y-axis center value [pulses] = -8388608–8388607

## Remarks

This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help: **F176_PulseOutput_Pass**

Use **PulseInfo_IsActive** to check if the control flag for the selected channel is FALSE.

*Example*

## DUT

With a Data Unit Type (DUT) you can define a data unit type that is composed of other data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

PulseOutput_Cha...tion_DUT [DUT]

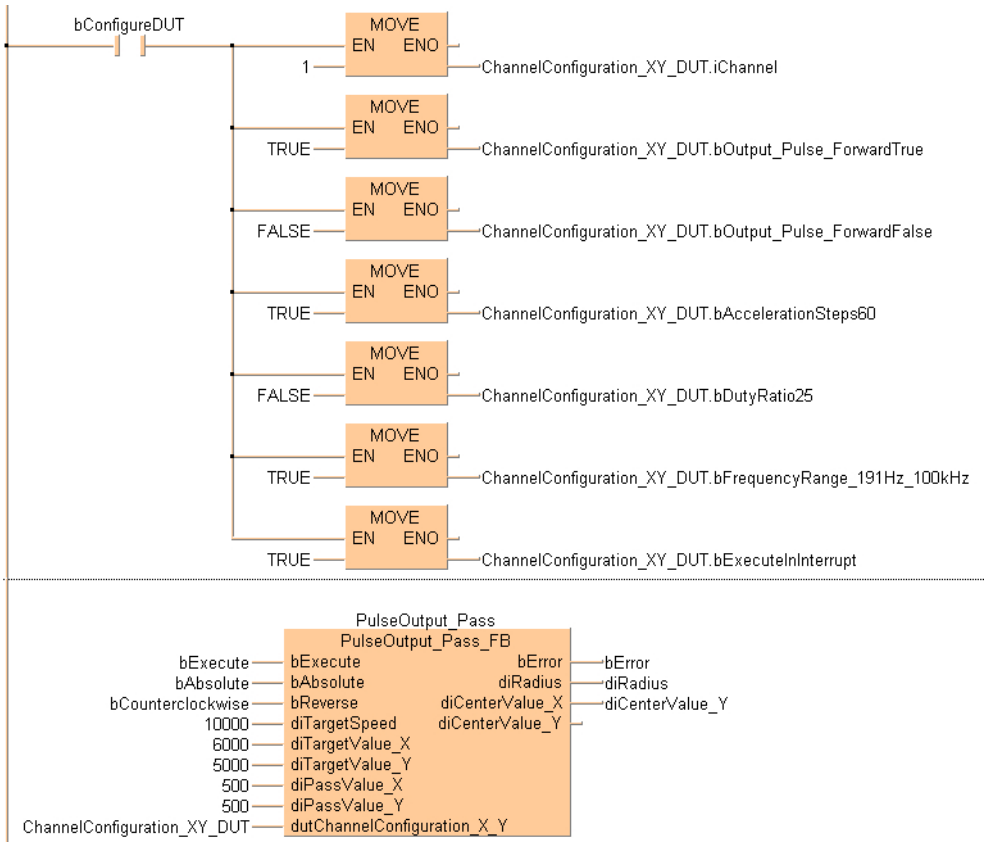
| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | iChannel | INT | 0 | FP-SIGMA: 0, 2 |
| 1 | bOutput_Pulse_ForwardTrue | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 2 | bOutput_Pulse_ForwardFalse | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 3 | bAccelerationSteps60 | BOOL | TRUE | FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps) |
| 4 | bDutyRatio25 | BOOL | TRUE | FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%) |
| 5 | bFrequencyRange_48Hz_100kHz | BOOL | FALSE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz |
| 6 | bFrequencyRange_191Hz_100kHz | BOOL | TRUE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz |
| 7 | bPulseWidth80µs | BOOL | FALSE | FP0, FP-e Home, Trapezoidal: 80µs (else 50%) |
| 8 | iDutyRatioIn10PercentSteps | INT | 0 | FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs |
| 9 | bEnableHomeOnlyAfterNearHomeDeceleration | BOOL | FALSE | FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1) |
| 10 | iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms | INT | 0 | FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms] |
| 11 | bCalculationOnly | BOOL | FALSE | FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output) |
| 12 | bTrapezoidalMaximumTargetSpeed50kHz | BOOL | FALSE | FP0R: Output operation: Type 1: The target speed can be up to the maximum s... |
| 13 | bExecuteInInterrupt | BOOL | FALSE | FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (... |
| 14 | bJogWithNoCounting | BOOL | FALSE | Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b... |
| 15 | bContinueAfterDone | BOOL | FALSE | FP-SIGMA circular pulse output: 0=Execution stops when target value has been... |

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | PulseOutput_Pass | PulseOutput_Pass_FB | |
| 1 | VAR | bExecute | BOOL | FALSE |
| 2 | VAR | bAbsolute | BOOL | FALSE |
| 3 | VAR | bContinueAfterDone | BOOL | FALSE |
| 4 | VAR | bCounterclockwise | BOOL | FALSE |
| 5 | VAR | ChannelConfiguration_XY_DUT | PulseOutput_Channel_Configuration_DUT | |
| 6 | VAR | bError | BOOL | FALSE |
| 7 | VAR | diRadius | DINT | 0 |
| 8 | VAR | diCenterValue_X | DINT | 0 |
| 9 | VAR | diCenterValue_Y | DINT | 0 |
| 10 | VAR | bConfigureDUT | BOOL | FALSE |

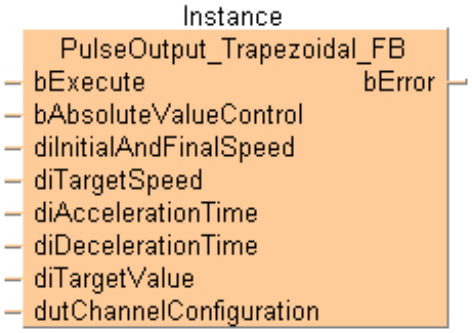
## LD body


bConfigureDUT
MOVE
EN ENO
1
ChannelConfiguration_XY_DUT.iChannel
TRUE
ChannelConfiguration_XY_DUT.bOutput_Pulse_ForwardTrue
FALSE
ChannelConfiguration_XY_DUT.bOutput_Pulse_ForwardFalse
ChannelConfiguration_XY_DUT.bAccelerationSteps60
ChannelConfiguration_XY_DUT.bDutyRatio25
ChannelConfiguration_XY_DUT.bFrequencyRange_191Hz_100kHz
ChannelConfiguration_XY_DUT.bExecuteInInterrupt
PulseOutput_Pass
PulseOutput_Pass_FB
bExecute
bAbsolute
bCounterclockwise
10000
6000
5000
500
500
ChannelConfiguration_XY_DUT
bReverse
diTargetSpeed
diTargetValue_X
diTargetValue_Y
diPassValue_X
diPassValue_Y
dutChannelConfiguration_X_Y
bError
diRadius
diCenterValue_X
diCenterValue_Y

# PulseOutput_Trapezoidal_FB

Trapezoidal control

This instruction automatically performs trapezoidal control according to the parameters in the function block and in the specified DUT. Pulses are output from the specified channel when the control flag for this channel is FALSE and the execution condition is TRUE.




## Parameters

Input

**bExecute** (BOOL)

FP-SIGMA, FP-X, FP0, **F168_PulseOutput_Trapezoidal**: Only with edge trigger
**F171_PulseOutput_Trapezoidal**: Execution condition can be:with edge triggerpermanent, if change of speed is required.

**bAbsoluteValueControl** (BOOL)

Absolute value control = TRUE, Relative value control = FALSE

**diInitialAndFinalSpeed** (DINT)

Initial and final speed: Set this value according to the frequency range selected in **PulseOutput_Channel_Configuration_DUT**:
FPΣ, FP-X: 1–9800 (1.5Hz–9.8kHz)
48–100000 (48Hz–100kHz)
191–100000 (191–100kHz)
**F171_PulseOutput_Trapezoidal**: 1–50000 (1Hz–50kHz)
FP0, **F168_PulseOutput_Trapezoidal**: 40–5000 (40Hz–5kHz)

**diTargetSpeed** (DINT)

Target speed: Set this value according to the frequency range selected in **PulseOutput_Channel_Configuration_DUT**:
FPΣ, FP-X: 1–9800 (1.5Hz–9.8kHz)
48–100000 (48Hz–100kHz)

191–100000 (191–100kHz)
**F171_PulseOutput_Trapezoidal**: 1–50000 (1Hz–50kHz)
FP0, **F168_PulseOutput_Trapezoidal**: 40–5000 (40Hz–5kHz)

**diAccelerationTime** (DINT)

Acceleration/deceleration time (FPΣ, FP-X):
With 30 steps: 30ms–32760ms (specify in steps of 30)
With 60 steps: 60ms–32760ms (specify in steps of 60)
Acceleration/deceleration time (FP0, **F168_PulseOutput_Trapezoidal**): 30ms–32760ms
Acceleration time (**F171_PulseOutput_Trapezoidal**): 1ms–32760ms

**diDecelerationTime** (DINT)

Deceleration time (**F171_PulseOutput_Trapezoidal**): 1ms–32760ms

**diTargetValue** (DINT)

Target value[pulses]: -2147483648–2147483647

**dutChannelConfiguration**Predefined system DUT for channel configuration: **PulseOutput_Channel_Configuration_DUT**

Output

**bError** (BOOL)

TRUE if an applied input value is invalid. Execution of the function block stops.

## Remarks

This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help:

- FPΣ, FP-X: **F171_PulseOutput_Trapezoidal**
- FP0: **F168_PulseOutput_Trapezoidal**

Use **PulseInfo_IsActive** to check if the control flag for the selected channel is FALSE.

*Example*

## DUT

With a Data Unit Type (DUT) you can define a data unit type that is composed of other data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

PulseOutput_Cha...tion_DUT [DUT]

| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | iChannel | INT | 0 | FP-SIGMA: 0, 2 |
| 1 | bOutput_Pulse_ForwardTrue | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 2 | bOutput_Pulse_ForwardFalse | BOOL | FALSE | if neither bOutput_Pulse_ForwardTrue nor bOutput_Pulse_ForwardFalse: Forw... |
| 3 | bAccelerationSteps60 | BOOL | TRUE | FP-SIGMA, FP-X: Number of acceleration/deceleration steps: 60 (else 30 steps) |
| 4 | bDutyRatio25 | BOOL | TRUE | FP-SIGMA, FP-X: Duty ratio (for pulse duration and period): 25% (else 50%) |
| 5 | bFrequencyRange_48Hz_100kHz | BOOL | FALSE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 48Hz–100kHz |
| 6 | bFrequencyRange_191Hz_100kHz | BOOL | TRUE | FP-SIGMA, FP-X: Frequency range for initial and target speed: 191Hz–100kHz |
| 7 | bPulseWidth80µs | BOOL | FALSE | FP0, FP-e Home, Trapezoidal: 80µs (else 50%) |
| 8 | iDutyRatioIn10PercentSteps | INT | 0 | FP0, FP-e Jog: 1-9: duty ratio in increments of 10%, 0: fixed pulse width of 80µs |
| 9 | bEnableHomeOnlyAfterNearHomeDeceleration | BOOL | FALSE | FP0R: Type 1 (else type 0), FP-SIGMA, FP-X: Type 2 (else type 1) |
| 10 | iHomeInputDeviationCounterClearSignalOutputTime_in_0p5ms | INT | 0 | FP0R, FP-SIGMA, FP-X: 0 to 200 [x0.5ms] |
| 11 | bCalculationOnly | BOOL | FALSE | FP0R: Jog, Trapezoidal: Output operation calculation only (else pulse output) |
| 12 | bTrapezoidalMaximumTargetSpeed50kHz | BOOL | FALSE | FP0R: Output operation: Type 1: The target speed can be up to the maximum s... |
| 13 | bExecuteInInterrupt | BOOL | FALSE | FP0R Jog positioning, trapezoidal: Execute in or called from interrupt program (... |
| 14 | bJogWithNoCounting | BOOL | FALSE | Only pulse outputs without counting, no target value match. FP-SIGMA, FP-X: b... |
| 15 | bContinueAfterDone | BOOL | FALSE | FP-SIGMA circular pulse output: 0=Execution stops when target value has been... |

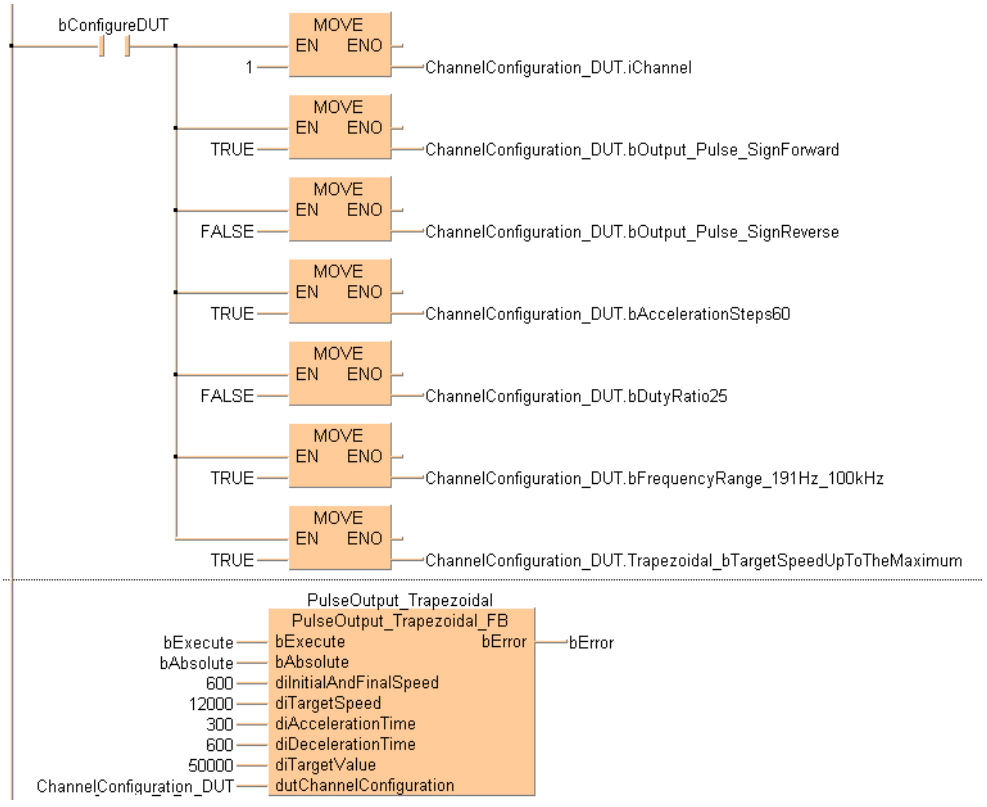
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | PulseOutput_Trapezoidal | PulseOutput_Trapezoidal_FB | |
| 1 | VAR | bExecute | BOOL | FALSE |
| 2 | VAR | bAbsolute | BOOL | FALSE |
| 3 | VAR | ChannelConfiguration_DUT | PulseOutput_Channel_Configuration_DUT | |
| 4 | VAR | bError | BOOL | FALSE |
| 5 | VAR | bConfigureDUT | BOOL | FALSE |

## POU body

## LD body

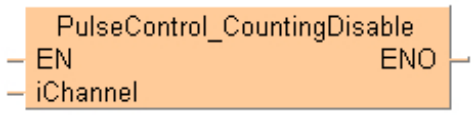


### 26.3.5.3 Pulse control instructions

# PulseControl_CountingDisable

Disables counting on pulse output channel

This instruction disables counting on the channel specified by **iChannel**. Bit 1 of the pulse output control code is set to TRUE.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**ENO** (BOOL)

TRUE if pulse counting has been disabled

*Example*

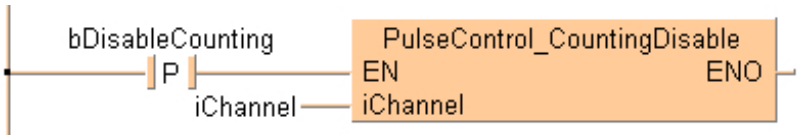
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bDisableCounting | BOOL | FALSE |

## POU body

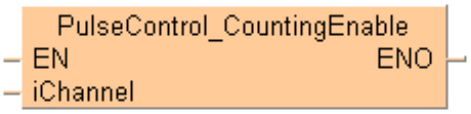
**LD body**


bDisableCounting
P
iChannel
PulseControl_CountingDisable
EN
ENO
iChannel

# PulseControl_CountingEnable

Enables counting on pulse output channel

This instruction enables counting on the pulse output channel specified by **iChannel** after counting has been disabled with **PulseControl_CountingDisable**. Bit 1 of the pulse output control code is set to FALSE.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

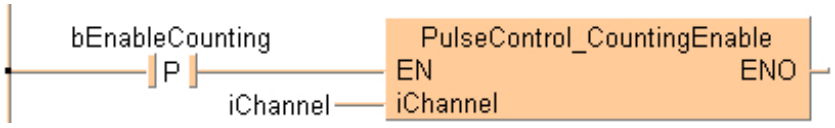
*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bEnableCounting | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

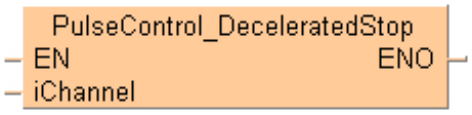
## LD body



# PulseControl_DeceleratedStop

Performs decelerated stop

This instruction performs a decelerated stop on the channel specified by **iChannel**. When a decelerated stop is requested during acceleration, deceleration is performed with the same slope as deceleration from the target speed.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

*Example*

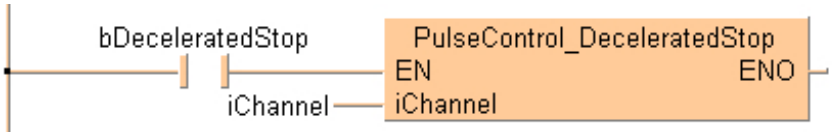
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bDeceleratedStop | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

## POU body

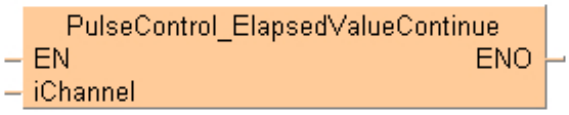
### LD body


bDeceleratedStop
PulseControl_DeceleratedStop
EN
ENO
iChannel
iChannel

# PulseControl_ElapsedValueContinue

Continues pulse counting after reset

This instruction resumes pulse counting on the channel specified by **PulseControl_ElapsedValueReset** after a reset of the elapsed value using **PulseControl_ElapsedValueReset**. Bit 0 of the pulse output control code is set to FALSE.




Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

*Example*

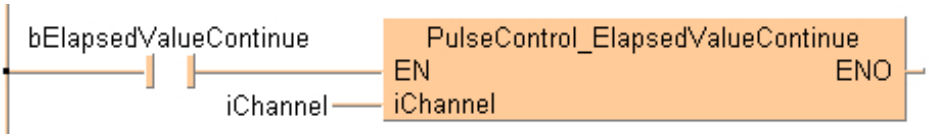
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bElapsedValueContinue | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

## POU body

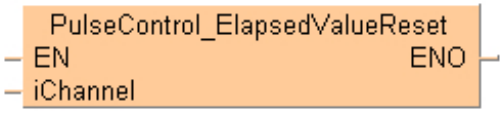
### LD body


bElapsedValueContinue
PulseControl_ElapsedValueContinue
EN
ENO
iChannel
iChannel

# PulseControl_ElapsedValueReset

Sets elapsed value to 0

This instruction sets the elapsed value of the pulse output channel specified by iChannel to 0. Bit 0 of the pulse output control code is set to TRUE. Use **PulseInfo_IsElapsedValueReset** to continue counting on the pulse output channel. Use **PulseInfo_IsElapsedValueReset** to check the current state. Pulse output continues when resetting the elapsed value.




Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

*Example*

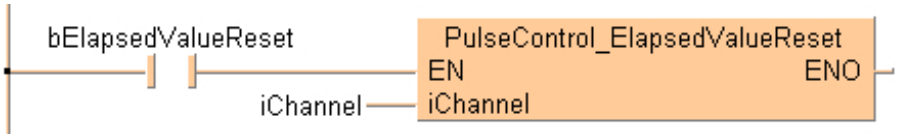
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bElapsedValueReset | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

## POU body

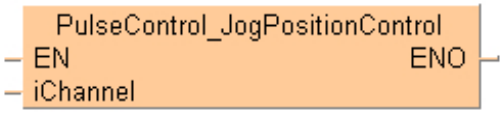
### LD body


bElapsedValueReset
PulseControl_ElapsedValueReset
EN
ENO
iChannel
iChannel

# PulseControl_JogPositionControl

Starts position control

This instruction sets and resets bit 6 of the pulse output control code to start position control on the channel specified by **iChannel**. The position control trigger is used with the JOG operation instructions **PulseOutput_Jog_Positioning0_FB** and **PulseOutput_Jog_Positioning1_FB**.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

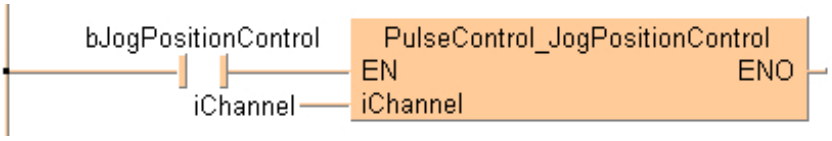
*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bJogPositionControl | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

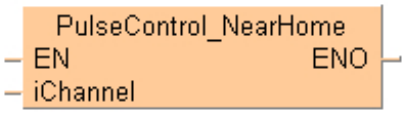
### LD body


bJogPositionControl
iChannel
PulseControl_JogPositionControl
EN
ENO
iChannel

# PulseControl_NearHome

Starts deceleration when near home

This instruction starts deceleration on the channel specified by **iChannel** when near the home input by setting bit 4 of the pulse output control code to TRUE and back to FALSE again. Use PulseInfo_IsHomeInputTrue to check if the home input is TRUE.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

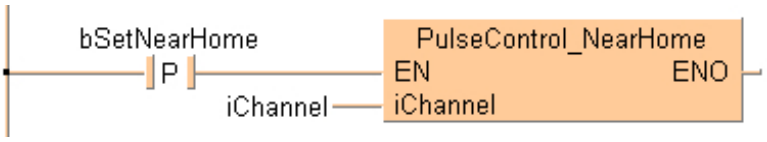
*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetNearHome | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

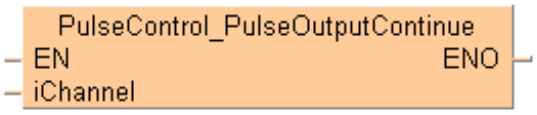
### LD body


bSetNearHome
P
iChannel
PulseControl_NearHome
EN
ENO
iChannel

# PulseControl_PulseOutputContinue

Continues pulse output

This instruction continues pulse output at the channel specified by **PulseControl_PulseOutputStop** after pulse output has been stopped using **PulseControl_PulseOutputStop**. Bit 3 of the pulse output control code is set to FALSE.




Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

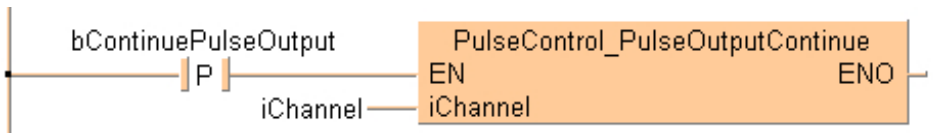
*Example*

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bContinuePulseOutput | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

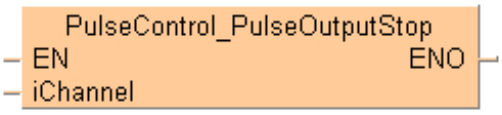
### LD body



# PulseControl_PulseOutputStop

Stops pulse output

This instruction stops the pulse output on the channel specified by **iChannel** by setting bit 3 of the pulse output control code to TRUE. Use **PulseControl_PulseOutputContinue** to continue pulse output after the interrupt.




Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

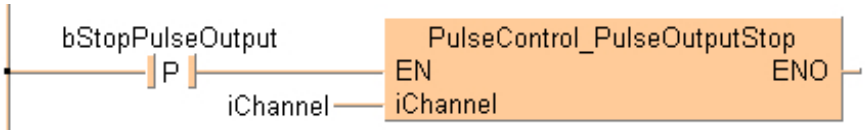
*Example*

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStopPulseOutput | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

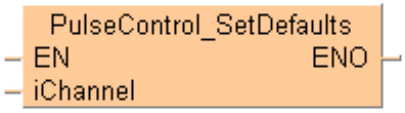
### LD body



# PulseControl_SetDefaults

Sets defaults for pulse output channel

This instruction sets all bits of the pulse output control code of the channel specified by **iChannel** to 0. 0 is the default setting.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

*Example*

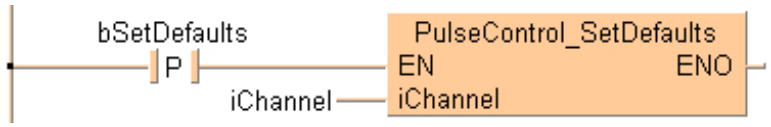
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bSetDefaults | BOOL | FALSE |
| 1 | VAR | iChannel | INT | 0 |

## POU body

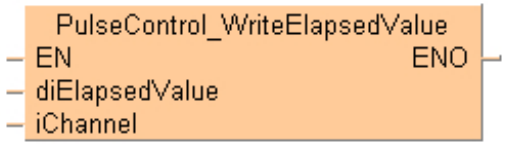
### LD body


bSetDefaults
P
iChannel
PulseControl_SetDefaults
EN
ENO
iChannel

# PulseControl_WriteElapsedValue

Writes elapsed value into pulse output channel

This instruction writes an elapsed value into pulse output channel specified by **iChannel**.




## Parameters

Input

**diElapsedValue** (DINT)

Elapsed value to be written into the channel specified by **iChannel**

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

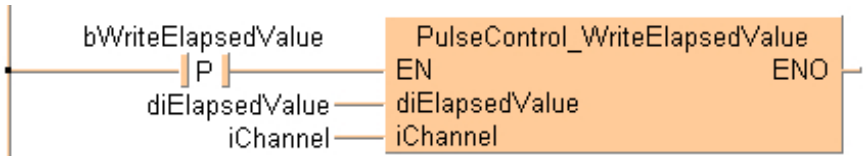
*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bWriteElapsedValue | BOOL | FALSE |
| 1 | VAR | diElapsedValue | DINT | 5000 |
| 2 | VAR | iChannel | INT | 0 |

### LD body


bWriteElapsedValue
P
diElapsedValue
iChannel
PulseControl_WriteElapsedValue
EN
ENO
diElapsedValue
iChannel

# PulseControl_TargetValueMatchClear

Clears target value match control

This instruction clears the target value match control on the channel specified by **iChannel**.


## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

*Example*

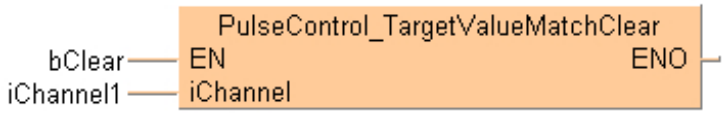
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel1 | INT | 1 |
| 1 | VAR | bClear | BOOL | FALSE |

## POU body

### LD body

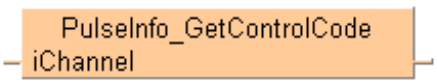

PulseControl_TargetValueMatchClear
bClear
EN
ENO
iChannel1
iChannel

### 26.3.5.4 Pulse information instructions

# PulseInfo_GetControlCode

Returns control code of pulse output channel

This instruction returns the control code of the pulse output channel specified by **iChannel**.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (WORD)

ResultStores the control code

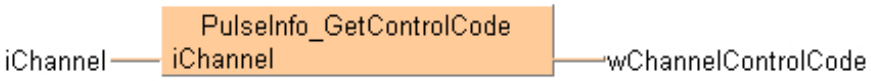
*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | wChannelControlCode | WORD | 0 |

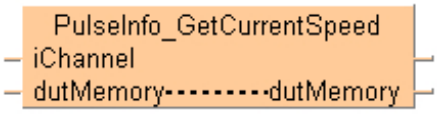
### LD body


PulseInfo_GetControlCode
iChannel
iChannel
wChannelControlCode

# PulseInfo_GetCurrentSpeed

Returns current pulse output frequency

This instruction returns the current speed in Hz of the pulse output channel specified by **iChannel**.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Input/output

**dutMemory** (PulseInfo_GetCurrentSpeed_DUT)

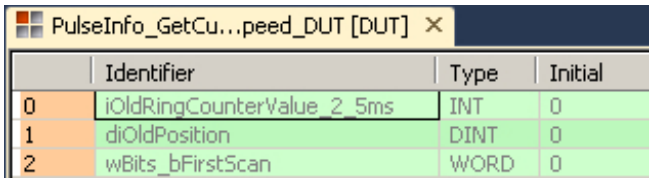
Output

**VAR_OUT** (DINT)

Result current pulse output frequency in Hz

*Example*

**DUT**

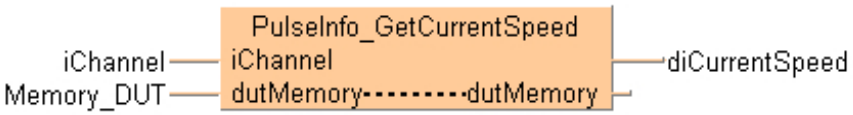


### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | Memory_DUT | PulseInfo_GetCurrentSpeed_DUT | |
| 2 | VAR | diCurrentSpeed | DINT | 0 |

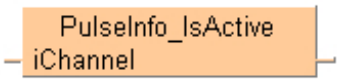
### LD body



# PulseInfo_IsActive

Check if pulse output is active

This instruction evaluates the pulse output control flag and returns TRUE if the pulse output channel specified by **iChannel** is active.




Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (WORD)

ResultTRUE if pulse output is active

*Example*

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bPulseOutputActive | BOOL | FALSE |

### LD body

# PulseInfo_IsChannelEnabled

Checks if pulse output channel is enabled

This instruction returns TRUE if the pulse output channel specified by **iChannel** has been enabled in the system registers and is supported by the selected PLC type.


## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (BOOL)

ResultTRUE if the pulse output channel specified by **iChannel** is enabled

*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bChannelEnabled | BOOL | FALSE |

### LD body


PulseInfo_IsChannelEnabled
iChannel
iChannel
bChannelEnabled

# PulseInfo_IsCountingDisabled

Checks if pulse counting is disabled

This instruction returns TRUE if counting on the channel specified by **iChannel** has been disabled.


## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (BOOL)

ResultTRUE if the pulse output channel specified by **iChannel** is disabled

*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bCountingDisabled | BOOL | FALSE |

### LD body


PulseInfo_IsCountingDisabled
iChannel
iChannel
bCountingDisabled

# PulseInfo_IsElapsedValueReset

Checks if elapsed value is set to 0

This instruction returns TRUE if the elapsed value of the pulse output channel specified by **iChannel** has been reset to 0.


## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (BOOL)

ResultTRUE if the channel specified by **iChannel** has been reset

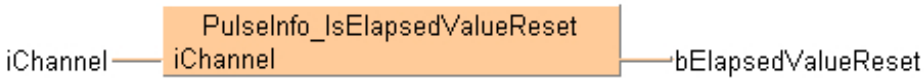
*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bElapsedValueReset | BOOL | FALSE |

### LD body


PulseInfo_IsElapsedValueReset
iChannel
iChannel
bElapsedValueReset

# PulseInfo_IsHomeInputTrue

Checks if home input is TRUE

This instruction returns TRUE if the home input of the channel specified by **iChannel** is TRUE.


## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (BOOL)

ResultTRUE if the home input has been reached

*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bHomeInput | BOOL | FALSE |

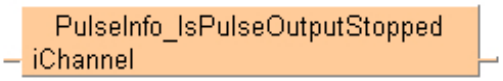
### LD body


PulseInfo_IsHomeInputTrue
iChannel
iChannel
bHomeInput

# PulseInfo_IsPulseOutputStopped

Checks if pulse output has stopped

This instruction returns TRUE if pulse output has been stopped, e.g. with **PulseControl_DeceleratedStop** or **PulseControl_PulseOutputStop**. Use **PulseControl_PulseOutputContinue** to resume pulse output.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (BOOL)

ResultTRUE if pulse output has been stopped

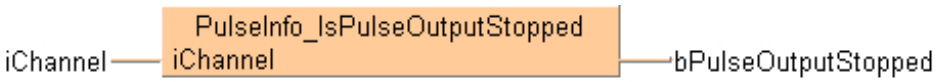
*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bPulseOutputStopped | BOOL | FALSE |

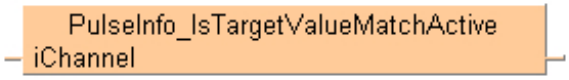
### LD body


PulseInfo_IsPulseOutputStopped
iChannel
iChannel
bPulseOutputStopped

# PulseInfo_IsTargetValueMatchActive

Checks if target value match control is active

This instruction returns TRUE if target value match control is active on the channel specified by **iChannel**.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (BOOL)

Result: TRUE if target value match control is active

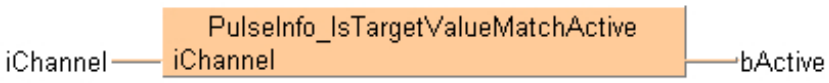
*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identi... | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | bActive | BOOL | FALSE |

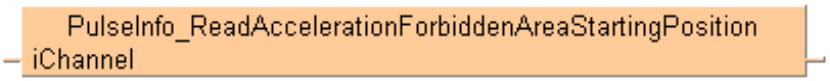
### LD body


PulseInfo_IsTargetValueMatchActive
iChannel
iChannel
bActive

# PulseInfo_ReadAccelerationForbiddenAreaStartingPosition

Read acceleration forbidden area starting position

This instruction reads the starting position of an acceleration forbidden area. If the elapsed value crosses over this position when the speed is being changed, acceleration cannot be continued any more.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (DINT)

Result stores the start position of the acceleration forbidden area

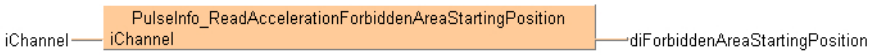
*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | diForbiddenAreaStartingPosition | DINT | 0 |

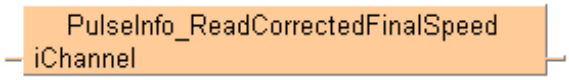
## LD body


PulseInfo_ReadAccelerationForbiddenAreaStartingPosition
iChannel
iChannel
diForbiddenAreaStartingPosition

# PulseInfo_ReadCorrectedFinalSpeed

Reads corrected value of final speed

This instruction returns the value of the corrected final speed on the channel specified by **iChannel**.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (INT)

Result stores the value of the corrected final speed

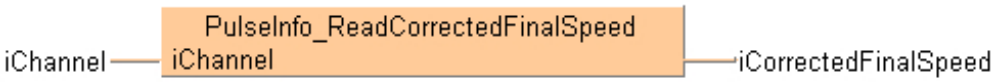
*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | iCorrectedFinalSpeed | INT | 0 |

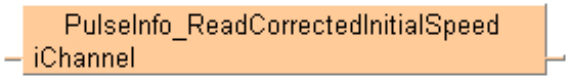
### LD body


PulseInfo_ReadCorrectedFinalSpeed
iChannel
iChannel
iCorrectedFinalSpeed

# PulseInfo_ReadCorrectedInitialSpeed

Reads corrected value of initial speed

This instruction returns the value of the corrected initial speed on the channel specified by **iChannel**.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (INT)

Resultstores the value of the corrected initial speed

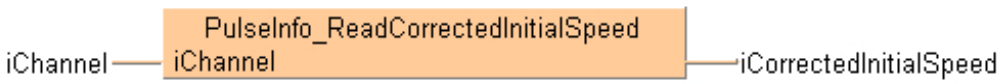
## *Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | iCorrectedInitialSpeed | INT | 0 |

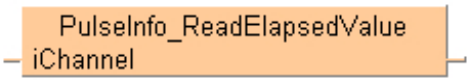
### LD body


PulseInfo_ReadCorrectedInitialSpeed
iChannel
iChannel
iCorrectedInitialSpeed

# PulseInfo_ReadElapsedValue

Reads elapsed value from pulse output channel

This instruction reads the elapsed value from the pulse output channel specified by **PulseControl_WriteElapsedValue**. Use **PulseControl_WriteElapsedValue** to modify the elapsed value and **PulseControl_ElapsedValueReset** to set the elapsed value to 0.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (DINT)

Resultstores the elapsed value from the channel specified by **iChannel**

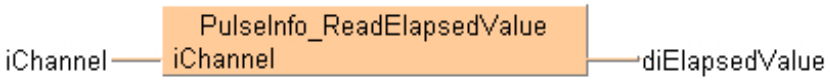
*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | diElapsedValue | DINT | 0 |

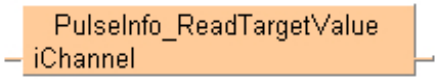
### LD body


PulseInfo_ReadElapsedValue
iChannel
iChannel
diElapsedValue

# PulseInfo_ReadTargetValue

Reads target value from pulse output channel

This instruction reads the target value from the pulse output channel specified by **iChannel**.




## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (DINT)

Resultstores the target value of the channel specified by **iChannel**

*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | diTargetValue | DINT | 0 |

### LD body


PulseInfo_ReadTargetValue
iChannel
iChannel
diTargetValue

# PulseInfo_ReadTargetValueMatchValue

Reads output control target value from pulse output channel

This instruction returns the output control target value of the pulse output channel specified by **iChannel**. The output control target value is used by the target value match instructions.


## Parameters

Input

**iChannel** (INT)

Pulse output channel:
FPΣ: 0, 2
FP-X/XH R: 0, 1
FP-X/XH 16K C14T: 0, 1, 2
FP-X/XH 32K C30T: 0, 1, 2, 3
FP-X/XH 32K C60T: 0, 1, 2, 3, 4, 5
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

Output

**VAR_OUT** (DINT)

Result stores the output control target value

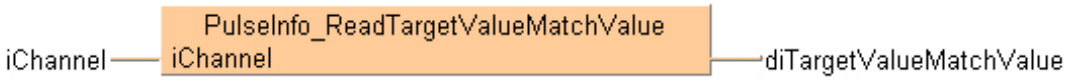
*Example*

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iChannel | INT | 0 |
| 1 | VAR | diTargetValueMatchValue | DINT | 0 |

### LD body

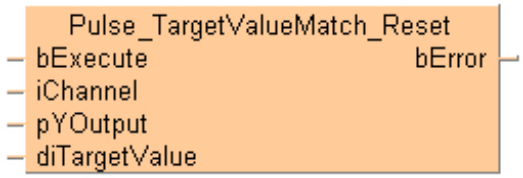

PulseInfo_ReadTargetValueMatchValue
iChannel
iChannel
diTargetValueMatchValue

### 26.3.5.5 Pulse output target value match control

# Pulse_TargetValueMatch_Reset

Target value match OFF (pulse output)

If the elapsed value matches the target value **diTargetValue** of the pulse output channel specified by **iChannel**, the output relay specified by **pYOutput** immediately turns to FALSE.




## Parameters

Input

**bExecute** (BOOL)

A rising edge activates the function; evaluate the pulse output channel control flag using PulseInfo_IsTargetValueMatchActive

**iChannel** (INT)

FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

**pYOutput** (POINTER)

Pointer result obtained by GetPointer from a global variable that supplies the channel number and output

**diTargetValue** (DINT)

Specify a 32-bit data value for the target value within the following range:
FP0, FP-e: -838808–+8388607
FPΣ, FP-X, FP0R: -2147483467–+2147483648

Output

**bError** (BOOL)

TRUE if the combination of `Channel%d` and `pYOuput.iOffset` does not match a valid combination of channel number and output as determined by the global variable

## Remarks

This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help:**F167_PulseOutput_Reset**

Note

To validate the combination of channel and Y output, the compiler requires the following name pattern for global variables:`%sPulse_TargetValueMatch_Channel%d_Y%d_%s`

Always use this pattern for global variables in target value match control.

- Channel%d must be a pulse output channel number enabled in the system registers
- `Y%d` must be an explicit output address supported by the PLC

| | |
|---|---|
| FP-Σ, FP0, FP-e: | Y0–Y7 |
| FP-Σ (V3.1 or higher), FP0R: | Y0–Y1F |
| FP-X: | Y0–Y29F |

- `%s` is an optional descriptor at the beginning of the pattern
- `_%s` is an optional descriptor at the end of the pattern

### Example

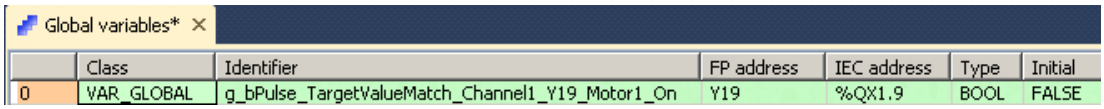
| Optional | Fixed pattern | Optional |
|---|---|---|
| `g_b` | `Pulse_TargetValueMatch_ChannelA_Y11F` | `_MotorOn` |

This global variable generates the code for channel **A** and output **Y11F**.

*Example*

## Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.



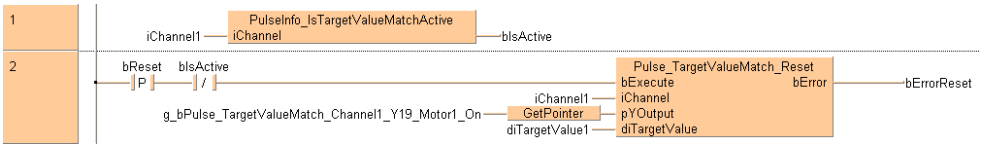

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_bPulse_TargetValueMatch_Channel1_Y19_Motor1_On | BOOL | FALSE |
| 1 | VAR | diTargetValue1 | DINT | 11000 |
| 2 | VAR | iChannel1 | INT | 1 |
| 3 | VAR | bIsActive | BOOL | FALSE |
| 4 | VAR | bErrorReset | BOOL | FALSE |
| 5 | VAR | bReset | BOOL | FALSE |

## LD body

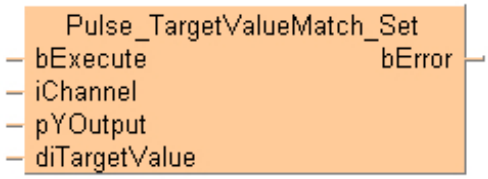
Use **PulseInfo_IsTargetValueMatchActive** to evaluate the channel **iChannel1** is active. If a rising edge is detected at **bReset** and if **bIsActive** is not TRUE, the instruction is executed. The combination of channel number and output contact is validated in the global variable **g_bPulse_TargetValueMatch_Channel1_Y19_Motor1_On**. When pulse output on channel 1 reaches the target value **diTargetValue1**, output Y19 is set to FALSE.



# Pulse_TargetValueMatch_Set

Target value match ON (pulse output)

If the elapsed value matches the target value **diTargetValue** of the pulse output channel specified by **iChannel**, the output relay specified by **pYOutput** immediately turns to TRUE.




## Parameters

Input

**bExecute** (BOOL)

A rising edge activates the function; evaluate the pulse output channel control flag using PulseInfo_IsTargetValueMatchActive

**iChannel** (INT)

FPΣ: 0, 2
FP-X R: 0, 1
FP-X 16K C14T: 0, 1, 2
FP-X 32K C30T, C60T: 0, 1, 2, 3
FP0R: 0, 1, 2, 3
FP0: 0, 1
FP-e: 0, 1

**pYOutput** (POINTER)

Pointer result obtained by GetPointer from a global variable that supplies the channel number and output

**diTargetValue** (DINT)

Specify a 32-bit data value for the target value within the following range:
FP0, FP-e: -838808–+8388607
FPΣ, FP-X, FP0R: -2147483467–+2147483648

Output

**bError** (BOOL)

TRUE if the combination of `Channel%d` and `pYOuput.iOffset` does not match a valid combination of channel number and output as determined by the global variable

### Remarks

This non-inline instruction is part of the tool instructions for pulse output. For a detailed description of the instruction(s) used internally, please refer to the online help:F166_PulseOutput_Set

Note

To validate the combination of channel and Y output, the compiler requires the following name pattern for global variables:`%sPulse_TargetValueMatch_Channel%d_Y%d_%s`

Always use this pattern for global variables in target value match control.

- Channel%d must be a pulse output channel number enabled in the system registers
- `Y%d` must be an explicit output address supported by the PLC

| | |
|---|---|
| FP-Σ, FP0, FP-e: | Y0–Y7 |
| FP-Σ (V3.1 or higher), FP0R: | Y0–Y1F |
| FP-X: | Y0–Y29F |

- `%s` is an optional descriptor at the beginning of the pattern
- `_%s` is an optional descriptor at the end of the pattern

### Example

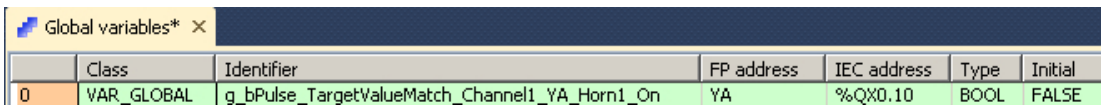
| | | |
|---|---|---|
| Optional | Fixed pattern | Optional |
| `g_b` | `Pulse_TargetValueMatch_ChannelA_Y11F` | `_MotorOn` |

This global variable generates the code for channel **A** and output **Y11F**.

*Example*

### Global variables

In the global variable list you define variables that can be accessed by all POUs in the project.



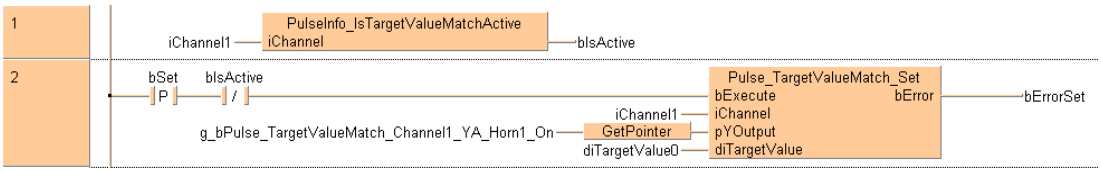

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR_EXTERNAL | g_bPulse_TargetValueMatch_Channel1_YA_Horn1_On | BOOL | FALSE |
| 1 | VAR | diTargetValue0 | DINT | 11000 |
| 2 | VAR | iChannel1 | INT | 1 |
| 3 | VAR | bIsActive | BOOL | FALSE |
| 4 | VAR | bErrorSet | BOOL | FALSE |
| 5 | VAR | bSet | BOOL | FALSE |

## LD body

Use **PulseInfo_IsTargetValueMatchActive** to evaluate the channel **iChannel1** is active. If a rising edge is detected at **bSet** and if **bIsActive** is not TRUE, the instruction is executed. The combination of channel number and output contact is validated in the global variable **g_bPulse_TargetValueMatch_Channel1_YA_Horn1_On**. When pulse output on channel 1 reaches the target value **diTargetValue0**, output YA is set to TRUE.



# 27 Selection instructions

# MAX

Maximum value

**MAX** determines the input variable with the highest value.


## Parameters

Input

**Unnamed input** (ANY) except STRING

1st input: value 1

**Unnamed input** (ANY) except STRING

2nd input: value 2

Output

**Unnamed output** all except STRING

result, whichever input variable's value is greater

## Remarks

This function can be expanded to a maximum of 28 input contacts, see also modifying elements.

# Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

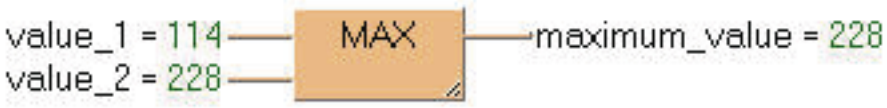
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | value_1 | INT | 0 | all types allowed |
| 1 | VAR | value_2 | INT | 0 | all types allowed |
| 2 | VAR | maximum_value | INT | 0 | all types allowed |

In this example the input variables (**value_1** and **value_2**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

### POU body

**Value_1** and **value_2** are compared with each other. The maximum value of all input variables is written in **maximum_value**.

### LD body




Note

This function can be expanded to a maximum of 28 input contacts, see also modifying elements.

# MIN

Minimum value

**MIN** detects the input variable with the lowest value.


## Parameters

Input

**Unnamed input** (ANY) except STRING

1st input: value 1

**Unnamed input** (ANY) except STRING

2nd input: value 2

Output

**Unnamed output** all except STRING

result, whichever input variable's value is smallest

## Remarks

This function can be expanded to a maximum of 28 input contacts, see also modifying elements.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | value_1 | INT | 0 | all types allowed |
| 1 | VAR | value_2 | INT | 0 | all types allowed |
| 2 | VAR | minimum_value | INT | 0 | all types allowed |

In this example the input variables (**value_1** and **value_2**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

### POU body

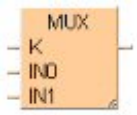
**Value_1** and **value_2** are compared with each other. The lower value of the two is written into **minimum_value**.

### LD body

# MUX

Select value from multiple channels

The function Multiplexer selects an input variable and writes its value into the output variable. The 1st input variable determines which input variable (**IN1**or **IN2** ...) is to be written into the output variable. The function **MUX** can be configured for any desired number of inputs.




## Parameters

Input

**K** (INT)

Selects which value is written to the output

**IN0** (ANY)

Value 0 is written to the output if **K** = 0

**IN1** (ANY)

Value 1 is written to the output if **K** = 1

Output

**VAR_OUT** (ANY)

Result output as 2nd and 3rd input

## Remarks

- When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.
- This function can be expanded to a maximum of 28 input contacts, see also modifying elements.
- The 2nd and 3rd input variables must be of the same data type.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | channel_select | INT | 0 | value '0' to 'n' |
| 1 | VAR | channel_0 | INT | 0 | all types allowed |
| 2 | VAR | channel_1 | INT | 0 | all types allowed |
| 3 | VAR | output | INT | 0 | all types allowed |

In this example the input variables (**channel_select**, **channel_0** and **channel_1**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

### POU body

In **channel_select** you find the integer value (0, 1...n) for the selection of **channel_0** or **channel_1**. The result will be written into **output**.

### LD body

# SEL

Select value from one of two channels

With the first input variable (data type BOOL) of **SEL** you define which input variable is to be written into the output variable. If the Boolean value = 0 (FALSE), the input variable **IN0** will be written into the output variable, otherwise **IN1**.


## Parameters

Input

**G** (BOOL)

1st input: selects between input value **IN0** or **IN1**

**IN0** (ANY)

2nd input: value is written into the output variable if **G** = FALSE

**IN1** (ANY)

3rd input: value is written into the output variable if **G** = TRUE

Output

**VAR_OUT** (ANY)

Result value as **IN0** or **IN1**

## Remarks

When using the data type STRING with small PLCs like FP-e or FP0, make sure that the length of the result string is equal to or greater than the length of the source string.

# Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

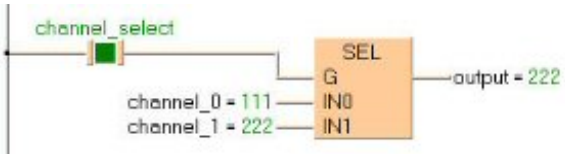
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | channel_select | BOOL | FALSE |
| 1 | VAR | channel_0 | INT | 0 |
| 2 | VAR | channel_1 | INT | 0 |
| 3 | VAR | output | INT | 0 |

In this example the input variables (**channel_select**, **channel_0** and **channel_1**) have been declared. Instead, you may enter a constant directly at the input contact of a function.

## POU body

If **channel_select** has the value 0, **channel_0** will be written into output, otherwise **channel_1**.

## LD body



# 28 SFC control instructions

## 28.1 Instructions that control all SFC programs simultaneously

# StartStopAllSfcs

Stop and restart all SFC programs

## Parameters

Input

**StopAllSfcs** (BOOL)

At a rising edge all SFC programs are stopped, and all step flags and explicit Boolean variables that have been declared as non-holding variables in the action association list of a step are reset.

**StartAllStoppedSfcs** (BOOL)

At a rising edge, all stopped SFC programs are restarted. SFC programs that are already running are not effected.

Output

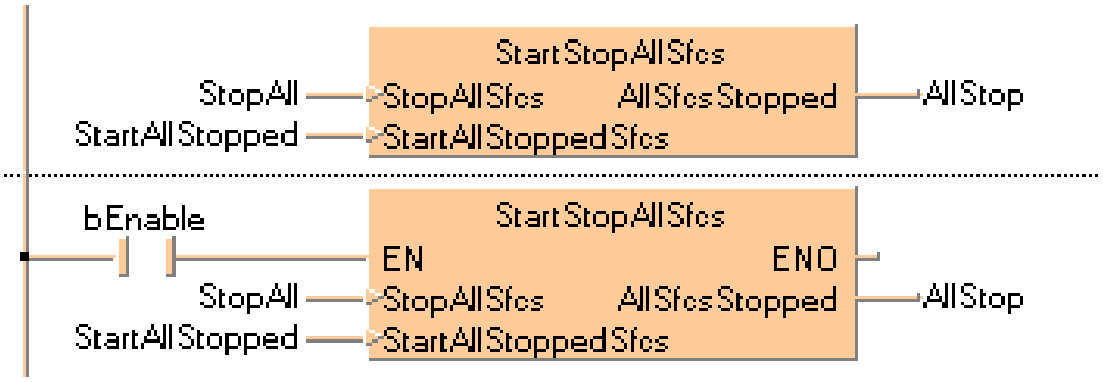
**AllSfcsStopped** (BOOL)

Set when all SFC programs are stopped. The function AllSfcsStopped produces this result, too.

## Remarks

- All explicit Boolean variables that have been set but not saved in the action association list of a step are reset. All other variables retain their last value.The stopped SFC programs can then be started all at once or individually with another SFC control function, or resumed at any position using **ActivateStepsOfStoppedSfcs**.
- This function cannot be used in functions because recognizing the rising requires a memory that a function does not have.
- This function cannot be used in SFCs.
- When this function is used, additional code is generated for the entire program. Since only 128 consecutive steps can be loaded to the PLC while in RUN mode, the code generated the first time this function is used cannot be loaded to the PLC. If you only use this function with the online edit mode in RUN mode, you have to download it at least once to the PLC.

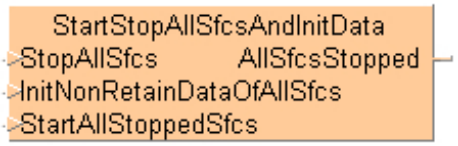
**Example**


StartStopAllSfcs
StopAll
StartAllStopped
StopAllSfcs
StartAllStoppedSfcs
AllSfcsStopped
AllStop
bEnable
StartStopAllSfcs
EN
ENO
StopAll
StartAllStopped
StopAllSfcs
StartAllStoppedSfcs
AllSfcsStopped
AllStop

# StartStopAllSfcsAndInitData

Stop and restart all SFC programs

With this function you can stop and restart all Sequential Function Chart (SFC) programs in a way that significantly saves program memory.Stopping the program means that all steps are deactivated and all step flags, e.g. **stepname.X**, reset.




## Parameters

Input

**StopAllSfcs** (BOOL)

At a rising edge all SFC programs are stopped, and all step flags and explicit Boolean variables that have been set but not saved in the action association list of a step are reset.

**InitNonRetainDataOfAllSfcs** (BOOL)

At a rising edge, all non-holding variables in the headers of the SFC program indicated by **SfcName**, including all external variables from the global variable list, are reinitialized. Explicit addresses used in an action or a transition are not affected.

**StartAllStoppedSfcs** (BOOL)

At a rising edge, all stopped SFC programs are restarted. SFC programs that are already running are not effected.

Output

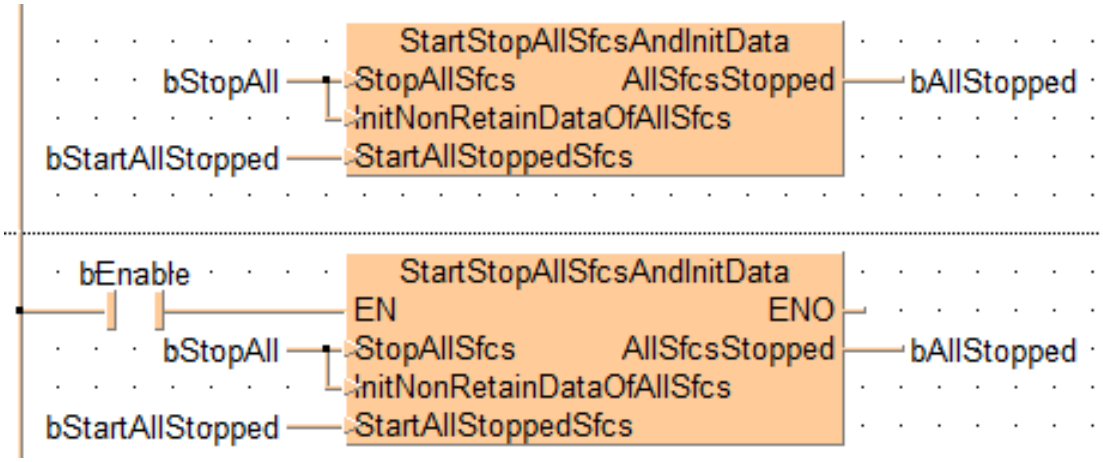
**AllSfcsStopped** (BOOL)

Indicates whether all SFC programs are stopped. The function **AllSfcsStopped** produces this result, too.

- All explicit Boolean variables that have been set but not saved in the action association list of a step are reset. All other variables retain their last value. In addition, the non-holding variables can be reinitialized. The stopped SFC programs can then be started all at once or individually with another SFC control function, or resumed at any position using **ActivateStepsOfStoppedSfc**.
- This function cannot be used in functions because recognizing the rising requires a memory that a function does not have.
- This function cannot be used in SFCs.

- When this function is used, additional code is generated for the entire program. Since only 128 consecutive steps can be loaded to the PLC while in RUN mode, the code generated the first time this function is used cannot be loaded to the PLC. If you only use this function with the online edit mode in RUN mode, you have to download it at least once to the PLC.

**Example**



## 28.2 A function that reveals the status of all SFCs

# AllSfcsStopped

Indicates whether all SFCs were stopped

The function indicates whether all Sequential Function Chart (SFC) programs were stopped, e.g. via **StartStopAllSfcs**.

## Parameters

Output

**VAR_OUT** (BOOL)

Set when all SFC programs are stopped.

## 28.3 Instructions that control a specific SFC

# StartStopSfc

Stop and restart a specific SFC program

With this function you can stop and restart a specific Sequential Function Chart (SFC) program.Stopping the program means that all steps are deactivated and all step flags, e.g. **stepname.X**, reset.

## Parameters

Input

**SfcName** (Literal)

Name of the SFC program to control

**Stop** (BOOL)

At a rising edge all SFC programs are stopped, and all step flags and explicit Boolean variables that have been set but not saved in the action association list of a step are reset.

**StartStoppedSfc** (BOOL)

At a rising edge, the stopped SFC program is restarted. An SFC program that is not stopped is not affected.

Output

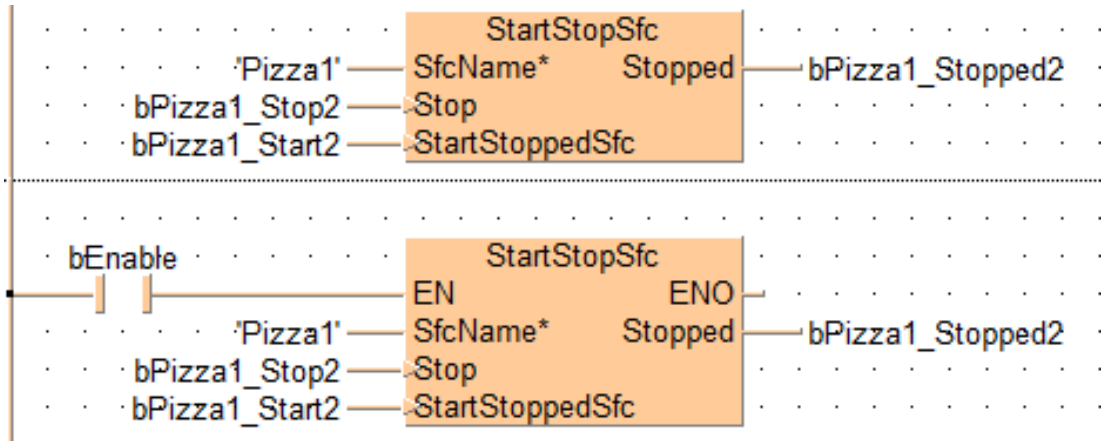
**Stopped** (BOOL)

Indicates whether the SFC program has been stopped. The function **SfcStopped** produces this result, too.

## Remarks

- All explicit Boolean variables that have been set but not saved in the action association list of a step are reset. All other variables retain their last value. The stopped SFC programs can then be started all at once or individually with another SFC control function, or resumed at any position using **ActivateStepsOfStoppedSfcs**.
- This function cannot be used in functions because recognizing the rising requires a memory that a function does not have.
- This function cannot be used in SFCs.
- When this function is used, additional code is generated for the entire program. Since only 128 consecutive steps can be loaded to the PLC while in RUN mode, the code generated the first time this function is used cannot be loaded to the PLC. If you only use this function with the online edit mode in RUN mode, you have to download it at least once to the PLC.

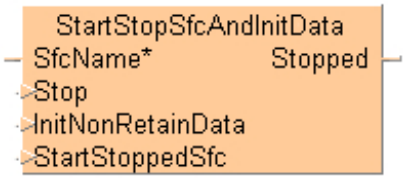
**Example**


StartStopSfc
'Pizza1'
SfcName*
Stopped
bPizza1_Stopped2
bPizza1_Stop2
Stop
bPizza1_Start2
StartStoppedSfc
bEnable
StartStopSfc
EN
ENO
'Pizza1'
SfcName*
Stopped
bPizza1_Stopped2
bPizza1_Stop2
Stop
bPizza1_Start2
StartStoppedSfc

# StartStopSfcAndInitData

Stop and restart a specific SFC program

With this function you can stop and restart a specific Sequential Function Chart (SFC) program.Stopping the program means that all steps are deactivated and all step flags, e.g. **stepname.X**, reset.




## Parameters

Input

**SfcName** (STRING)

Name of the SFC program to control

**Stop** (BOOL)

At a rising edge the SFC program is stopped and all step flags and explicit Boolean variables that have been set but not saved in the action association list of a step are reset.

**InitNonRetainData** (BOOL)

At a rising edge, all non-holding variables in the headers of the SFC program indicated by **SfcName**, including all external variables from the global variable list, are reinitialized. Explicit addresses used in an action or a transition are not affected.

**StartStoppedSfc** (BOOL)

At a rising edge, the stopped SFC program is restarted. An SFC program that is not stopped is not affected.

Output

**Stopped** (BOOL)

Indicates whether the SFC program has been stopped. The function **SfcStopped** produces this result, too.

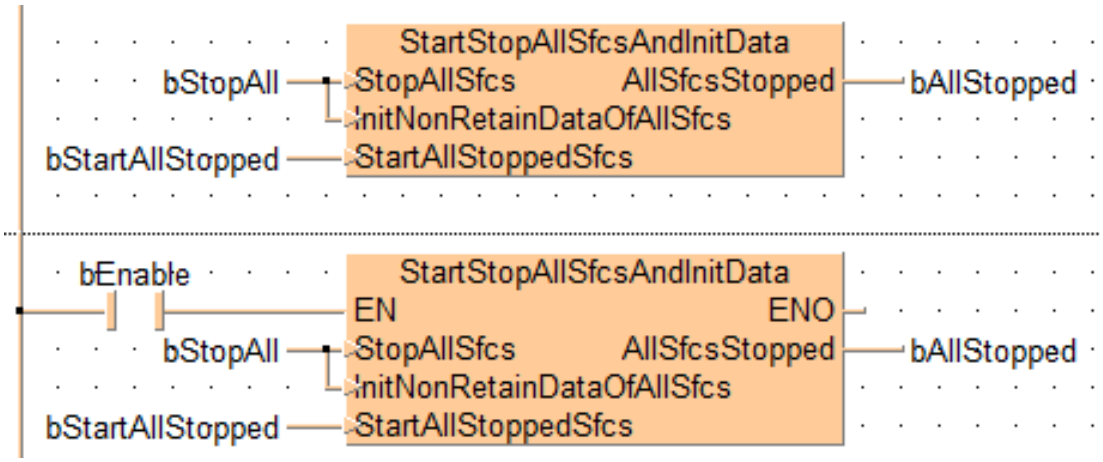
## Remarks

- All explicit Boolean variables that have been set but not saved in the action association list of a step are reset. All other variables retain their last value.In addition, the non-holding variables can be reinitialized. The stopped SFC programs can then be started all

at once or individually with another SFC control function, or resumed at any position using **ActivateStepsOfStoppedSfc**.

- This function cannot be used in functions because recognizing the rising requires a memory that a function does not have.
- This function cannot be used in SFCs.
- When this function is used, additional code is generated for the entire program. Since only 128 consecutive steps can be loaded to the PLC while in RUN mode, the code generated the first time this function is used cannot be loaded to the PLC. If you only use this function with the online edit mode in RUN mode, you have to download it at least once to the PLC.

### Example



# ControlSfc

Control a specific SFC program

With this function you can control a specific Sequential Function Chart (SFC) program. Stopping the program means that all steps are deactivated and all step flags, e.g. **stepname.X**, reset. Furthermore, you can lock all transition conditions, i.e. all transition conditions are always turned off. You can resume the program via the input **Run**.


## Parameters

Input

SfcName (Literal)

Name of the SFC program to control

Stop (BOOL)

The SFC program is stopped at a rising edge; all step flags and explicit Boolean variables that have been set but not saved in the action association list of a step are reset.

InhibitTransitions (BOOL)

All transitions are locked at a rising edge, i.e. the transition conditions are always turned off.

Run (BOOL)

At a rising edge, the SFC program that had been stopped is restarted and locked transitions are unlocked.

Output

Stopped (BOOL)

Indicates whether the SFC program has been stopped. The function **SfcStopped** produces this result, too.

TransitionsInhibited (BOOL)

Indicates whether the transitions are locked. The function **SfcTransitionsInhibited** produces this result, too.

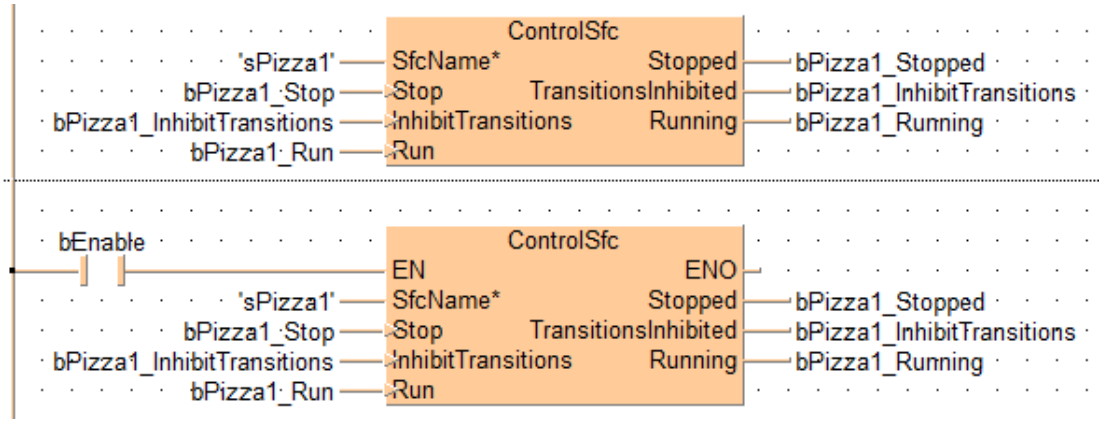
Running (BOOL)

Indicates whether the SFC program is running. The function **SfcRunning** produces this result, too.

## Remarks

- All explicit Boolean variables that have been set but not saved in the action association list of a step are reset. All other variables retain their last value.
- This function cannot be used in functions because recognizing the rising requires a memory that a function does not have.
- This function cannot be used in SFCs.
- When this function is used, additional code is generated for the entire program. Since only 128 consecutive steps can be loaded to the PLC while in RUN mode, the code generated the first time this function is used cannot be loaded to the PLC. If you only use this function with the online edit mode in RUN mode, you have to download it at least once to the PLC.

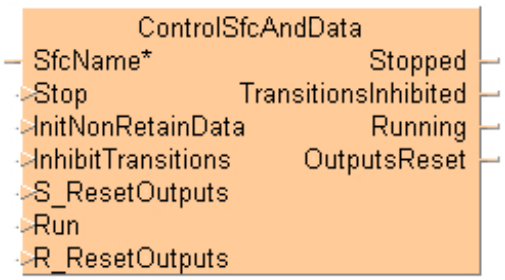
### Example



# ControlSfcAndData

Control a specific SFC program

With this function you can control a specific Sequential Function Chart (SFC) program. Stopping the program means that all steps are deactivated and all step flags, e.g. **stepname.X**, reset. Furthermore, you can lock all transition conditions, i.e. all transition conditions are always turned off. You can resume the program via the input **Run**.




## Parameters

Input

**SfcName** (STRING) (Literal)

Name of the SFC program to control

**Stop** (BOOL)

The SFC program is stopped at a rising edge; all step flags and explicit Boolean variables that have been set but not saved in the action association list of a step are reset.

**InitNonRetainData** (BOOL)

At a rising edge, all non-holding variables in the headers, including all external variables from the global variable list, are reinitialized. Explicit addresses used in an action or a transition are not effected.

**InhibitTransitions** (BOOL)

All transitions are locked at a rising edge, i.e. the transition conditions are always turned off.

**S_ResetOutputs** (BOOL)

At a rising edge a mode is set in which output variables in the address area Y are reset.

**Run** (BOOL)

At a rising edge, the SFC program that had been stopped is restarted and locked trasitions are unlocked.

**R_ResetOutputs**

At a rising edge the mode is reset and thus left in which output variables in the address area Y are reset.

Output

**Stopped** (BOOL)

Indicates whether the SFC program has been stopped. The function SfcStopped produces this result, too.

**TransitionsInhibited** (BOOL)

Indicates whether the transitions are locked. The function **SfcTransitionsInhibited** produces this result, too.

**Running** (BOOL)

Indicates whether the SFC program is running. The function **SfcRunning** produces this result, too.

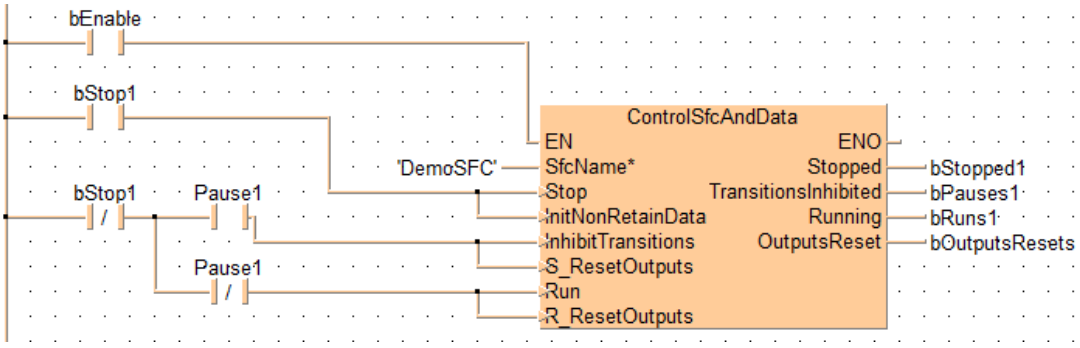
**OutputsReset** (BOOL)

Indicates whether the outputs are reset. The function SfcOutputsReset produces this result, too.

## Remarks

- All explicit Boolean variables that have been set but not saved in the action association list of a step are reset. All other variables retain their last value.
- This function cannot be used in functions because recognizing the rising requires a memory that a function does not have.
- This function cannot be used in SFCs.
- When this function is used, additional code is generated for the entire program. Since only 128 consecutive steps can be loaded to the PLC while in RUN mode, the code generated the first time this function is used cannot be loaded to the PLC. If you only use this function with the online edit mode in RUN mode, you have to download it at least once to the PLC.

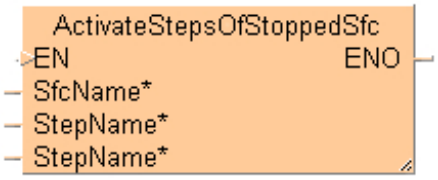
## Example



# ActivateStepsOfStoppedSfc

Continue a stopped SFC program at any position

With this expandable function you can continue a Sequential Function Chart (SFC) program that has been stopped at any position. This function is only carried out at a rising edge at the input.




## Parameters

Input

**SfcName** (Literal)

Name of the SFC program to control

**StepName** (Literal)

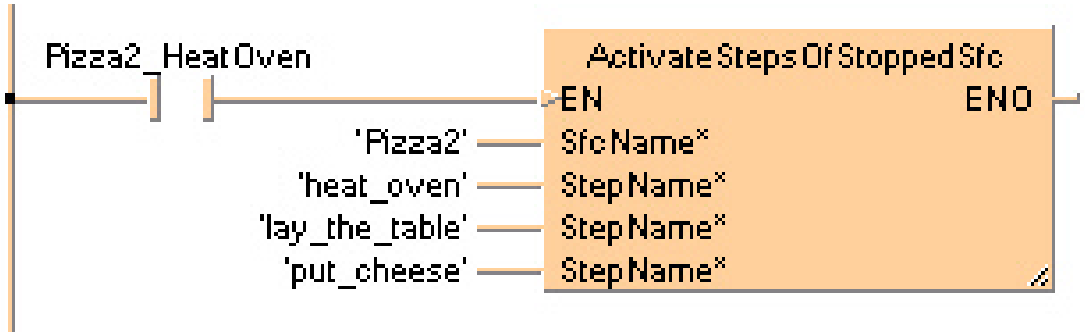
Expandable input that defines the step to be activated at the rising edge at the input.

## Remarks

- To stop an SFC program you can use one of the functions:
  - **StartStopAllSfcs** (page 1739)
  - **StartStopAllSfcsAndInitData** (page 1741)
  - **StartStopSfc** (page 1746)
  - **StartStopSfcAndInitData** (page 1748)
  - **ControlSfc** (page 1750)
  - **ControlSfcAndData** (page 1752)
- This function cannot be used in functions because recognizing the rising requires a memory that a function does not have.
- This function cannot be used in SFCs.
- When this function is used, additional code is generated for the entire program. Since only 128 consecutive steps can be loaded to the PLC while in RUN mode, the code generated the first time this function is used cannot be loaded to the PLC. If you only use this function with the online edit mode in RUN mode, you have to download it at least once to the PLC.
- This function is only carried out at a rising edge at the input.

- This function only checks whether the steps specified exist. It does not check whether the status of the SFC program is correct. Examples of incorrect states would be when more than one step is active in a simple sequence, when with an alternative divergence one step is active in more than one divergence, or when with a parallel divergence there is divergence without an active step.

**Example**



## 28.4 Instructions that reveal the statuses of a specific SFC

# SfcStopped

Indicates whether a specific SFC program was stopped

This function indicates whether a specific Sequential Function Chart (SFC) program was stopped, e.g. via **ControlSfc** or **ControlSfcAndData**.


## Parameters

Input

**SfcName** (Literal)

Name of the SFC program to control

Output

**Stopped** (BOOL)

Indicates whether the SFC program has been stopped.

# SfcTransitionsInhibited

Indicates whether the transitions of a specific SFC program are locked

This function indicates whether the transitions of a specific Sequential Function Chart (SFC) program are locked, e.g. via ControlSfc or ControlSfcAndData.


## Parameters

Input

**SfcName** (Literal)

Name of the SFC program to control

Output

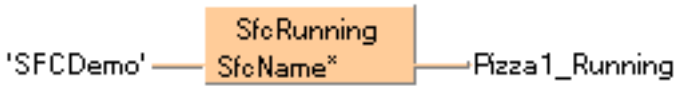
**TransitionsInhibited** (BOOL)

Indicates whether the transitions are locked.

# SfcRunning

Indicates whether a certain SFC program is running

This function indicates whether a certain Sequential Function Chart (SFC) program is running.




## Parameters

Input

**SfcName** (Literal)

Name of the SFC program to control

Output

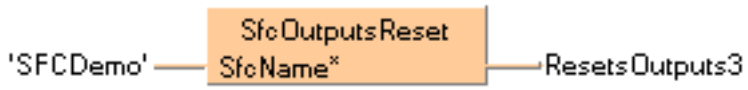
**Running** (BOOL)

Indicates whether the SFC program is running.

# SfcOutputsReset

Indicates whether the outputs of a SFC program have been reset

This function indicates whether the outputs of a specific Sequential Function Chart (SFC) program have been reset.




## Parameters

Input

**SfcName** (Literal)

Name of the SFC program to control

Output

**OutputsReset** (BOOL)

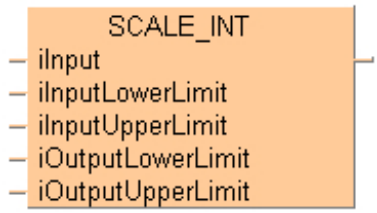
Indicates whether the outputs are reset.

# 29 Signal processing instructions

# SCALE_INT

Scale INTEGER data

This instruction scales an INTEGER value between a lower and an upper limit to an INTEGER output value. Use **WITHIN_LIMITS** to check if the input value is within the specified limits.




## Parameters

Input

**iInput** (INT)

Input signal

**iInputLowerLimit** (INT)

Lower limit of the input range

**iInputUpperLimit** (INT)

Upper limit of the input range

**iOutputLowerLimit** (INT)

Output value assigned to the lower limit of the input range (can be higher than **iOutputUpperLimit**)

**iOutputUpperLimit** (INT)

Output value assigned to the upper limit of the input range (can be lower than **iOutputLowerLimit**)

Output

**VAR_OUT** (INT)

Scaled output signal

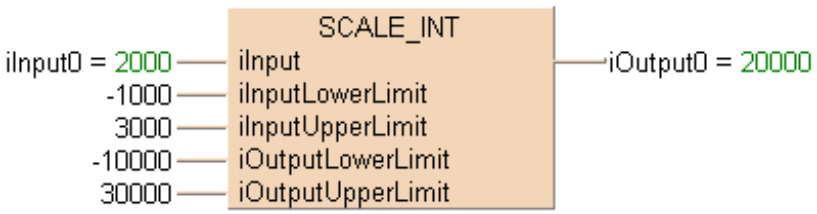
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iInput0 | INT | 2000 |
| 1 | VAR | iOutput0 | INT | 0 |

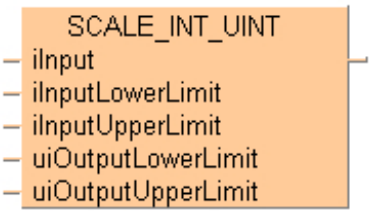
### LD body



# SCALE_INT_UINT

Scale INTEGER data to unsigned INTEGER data

This instruction scales an INTEGER value between a lower and an upper limit to an unsigned INTEGER output value. Use **WITHIN_LIMITS** to check if the input value is within the specified limits.




## Parameters

Input

**iInput** (INT)

Input signal

**iInputLowerLimit** (INT)

Lower limit of the input range

**iInputUpperLimit** (INT)

Upper limit of the input range

**uiOutputLowerLimit** (UINT)

Output value assigned to the lower limit of the input range (can be higher than **uiOutputUpperLimit**)

**uiOutputUpperLimit** (UINT)

Output value assigned to the upper limit of the input range (can be lower than **uiOutputLowerLimit**)

Output

**VAR_OUT** (UINT)

Scaled output signal

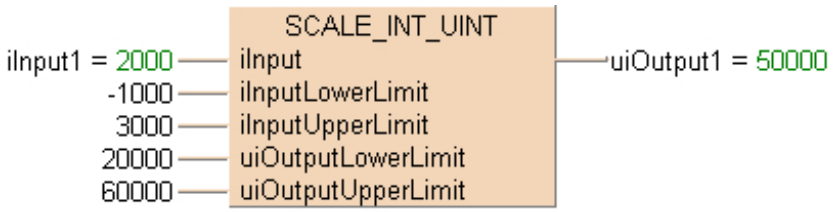
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iInput1 | INT | 2000 |
| 1 | VAR | uiOutput1 | UINT | 0 |

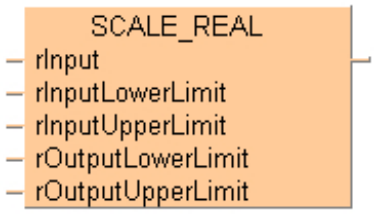
### LD body



# SCALE_REAL

Scale REAL data

This instruction scales a REAL value between a lower and an upper limit to a REAL output value. Use **WITHIN_LIMITS** to check if the input value is within the specified limits.




## Parameters

Input

**rInput** (REAL)

Input signal

**rInputLowerLimit** (REAL)

Lower limit of the input range

**rInputUpperLimit** (REAL)

Upper limit of the input range

**rOutputLowerLimit** (REAL)

Output value assigned to the lower limit of the input range (can be higher than **rOutputUpperLimit**)

**rOutputUpperLimit** (REAL)

Output value assigned to the upper limit of the input range (can be lower than **rOutputLowerLimit**)

Output

**VAR_OUT** (REAL)

Scaled output signal

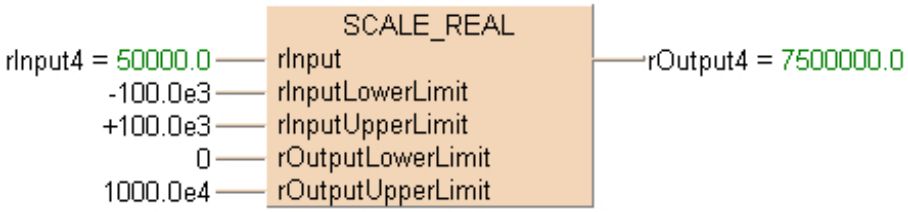
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | rInput4 | REAL | 50.0e3 |
| 1 | VAR | rOutput4 | REAL | 0.0 |

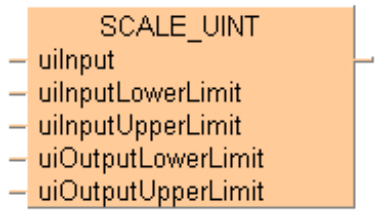
### LD body



# SCALE_UINT

Scale UINT data

This instruction scales an unsigned INTEGER value between a lower and an upper limit to an unsigned INTEGER output value. Use **WITHIN_LIMITS** to check if the input value is within the specified limits.




## Parameters

Input

**uiInput** (UINT)

Input signal

**uiInputLowerLimit** (UINT)

Lower limit of the input range

**uiInputUpperLimit** (UINT)

Upper limit of the input range

**uiOutputLowerLimit** (UINT)

Output value assigned to the lower limit of the input range (can be higher than **uiOutputUpperLimit**)

**uiOutputUpperLimit** (UINT)

Output value assigned to the upper limit of the input range (can be lower than **uiOutputLowerLimit**)

Output

**VAR_OUT** (UINT)

Scaled output signal

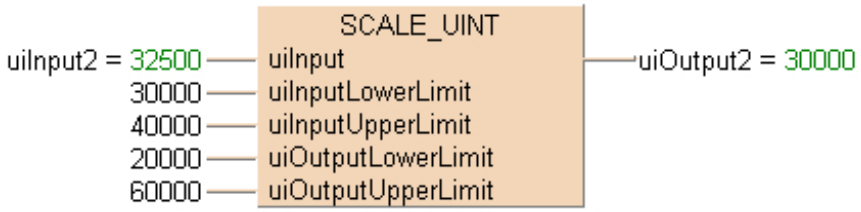
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | uiInput2 | UINT | 32500 |
| 1 | VAR | uiOutput2 | UINT | 0 |

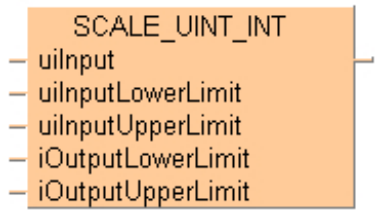
### LD body



# SCALE_UINT_INT

Scale UINT data to INT data

This instruction scales an unsigned INTEGER value between a lower and an upper limit to an INTEGER output value. Use **WITHIN_LIMITS** to check if the input value is within the specified limits.




## Parameters

Input

**uiInput** (UINT)

Input signal

**uiInputLowerLimit** (UINT)

Lower limit of the input range

**uiInputUpperLimit** (UINT)

Upper limit of the input range

**iOutputLowerLimit** (INT)

Output value assigned to the lower limit of the input range (can be higher than **iOutputUpperLimit**)

**iOutputUpperLimit** (INT)

Output value assigned to the upper limit of the input range (can be lower than **iOutputLowerLimit**)

Output

**VAR_OUT** (INT)

Scaled output signal

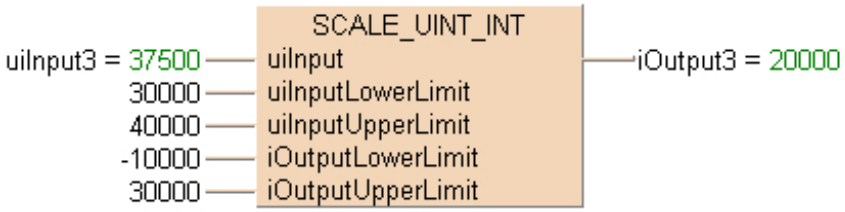
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | uiInput3 | UINT | 37500 |
| 1 | VAR | iInput3 | INT | 0 |

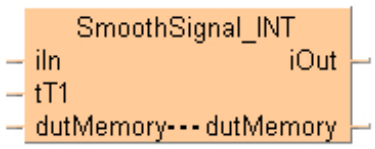
### LD body



# SmoothSignal_INT

Smooth INT signals

This instructions uses a 1st order delay time **tT1** to smooth the INTEGER input value at **iIN**.




## Parameters

Input

**iIn** (INT)

Input signal

**tT1** (TIME)

Time constant of the 1st order low-pass filter

Input/output

**dutMemory** (SmoothSignal_INT_DUT)

Instance-dependent data memory structure, which serves as the internal memory of the function. As with the instance name of a function block, it may be neither initialized nor written in the body!

Output

**iOut** (INT)

Output signal

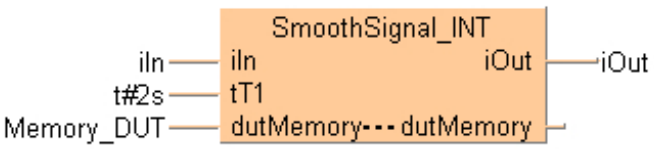
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | iIn | INT | 0 |
| 1 | VAR | iOut | INT | 0 |
| 2 | VAR | tT1 | TIME | T#0s |
| 3 | VAR | Memory_DUT | SmoothSignal_INT_DUT | |

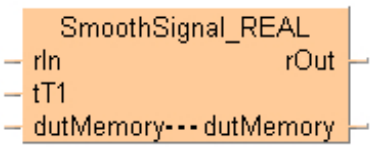
### LD body



# SmoothSignal_REAL

Smooth REAL signals

This instructions uses a 1st order delay time **tT1** to smooth the REAL input value at **iIN**.




## Parameters

Input

**rIn** (REAL)

Input signal

**tT1** (TIME)

Time constant of the 1st order low-pass filter

Input/output

**dutMemory** (SmoothSignal_REAL_DUT)

Instance-dependent data memory structure, which serves as the internal memory of the function. As with the instance name of a function block, it may be neither initialized nor written in the body!

Output

**rOut** (REAL)

Output signal

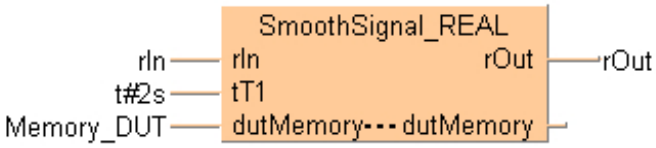
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | rIn | REAL | 0.0 |
| 1 | VAR | Memory_DUT | SmoothSignal_REAL_DUT | |
| 2 | VAR | rOut | REAL | 0.0 |

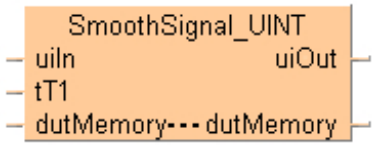
### LD body


SmoothSignal_REAL
rIn
t#2s
Memory_DUT
rIn
tT1
dutMemory
rOut
dutMemory
rOut

# SmoothSignal_UINT

Smooth UINT signals

This instructions uses a 1st order delay time **tT1** to smooth the unsigned INTEGER input value at **uiIn**.




## Parameters

Input

**uiIn** (UINT)

Input signal

**tT1** (TIME)

Time constant of the 1st order low-pass filter

Input/output

**dutMemory** (SmoothSignal_UINT_DUT)

Instance-dependent data memory structure, which serves as the internal memory of the function. As with the instance name of a function block, it may be neither initialized nor written in the body!

Output

**uiOut** (UINT)

Output signal

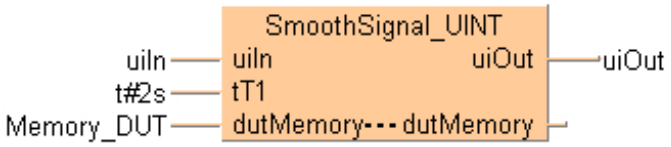
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | uiIn | UINT | 0 |
| 1 | VAR | Memory_DUT | SmoothSignal_UINT_DUT | |
| 2 | VAR | uiOut | UINT | 0 |

### LD body



## 29.9 FP instructions

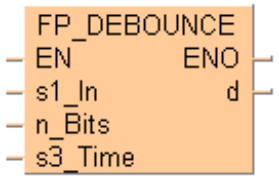
Tip

Advantages of FP instructions

# FP_DEBOUNCE

Debounce data by filtering bits over a specified time

This FP instruction executes filter processing for specified bits to ensure that their signal is stable for a specified time. The instruction can be useful to negate the effects of bounce, e.g. for a switching device.




## Parameters

Input

**s1_In** (WORD)

Input data whose bits will be filtered according to the input mask

**n_Bits** (WORD, INT, UINT)

Input mask which specifies which bits will be filtered

**s3_Time** (INT)

Specifies the time in ms during which the signal at the specified bits has to be stable

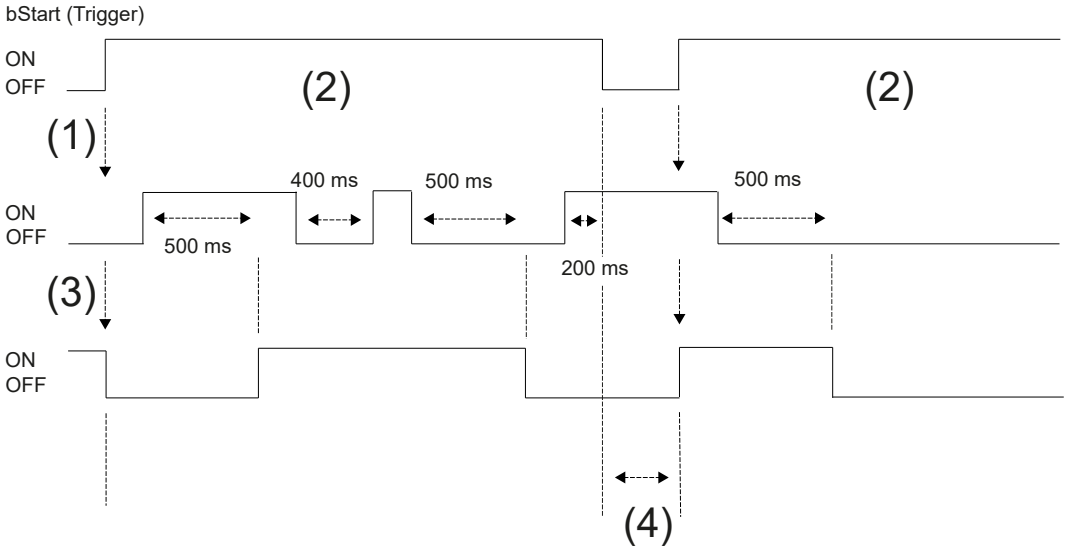
Output

**d** (WORD)

Filtered data

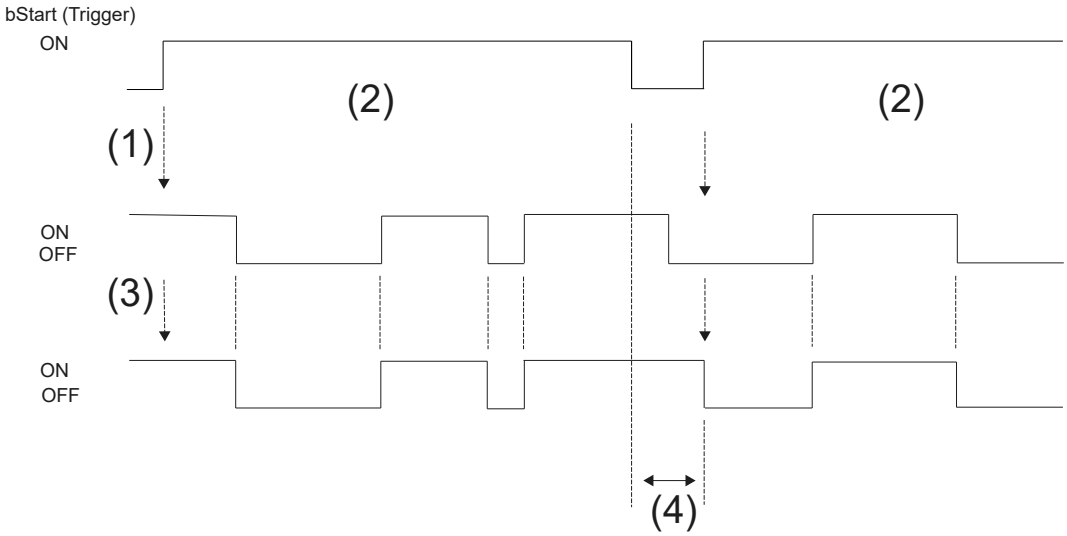
## Time chart

- 1 (16#0001) is assigned to **s2_InputMask** is , i.e. bit 0 will be filtered, the other bits will not be filtered, and the value assigned to **s3_FilterTime** is 500ms.




(1) Bit 0 of input data
(2) Bit 0 of filter result
(3) Data is initialized when the system detects the trigger's rising edge.
(4) The instruction does not operate while the trigger is OFF.

- 0 (16#0000) is assigned to **s2_InputMask** is , i.e. bit 0 to F will be not filtered




(1) Bit 1–15 of input data
(2) Bit 1–15 of filter result
(3) Data is initialized when the system detects the trigger's rising edge.
(4) The instruction does not operate while the trigger is OFF.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the filter processing time specified by **s3_Time** is less than 0 or greater than 30000.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the filter processing time specified by **s3_Time** is less than 0 or greater than 30000.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

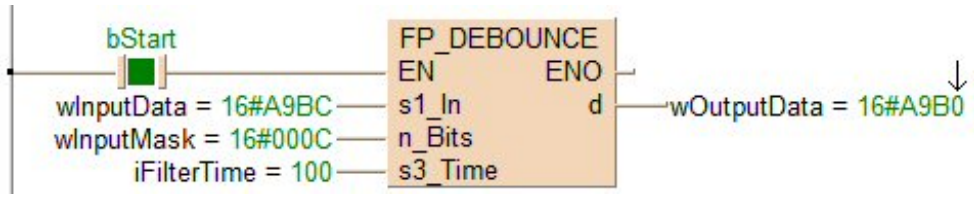
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | |
| 1 | VAR | wInputData | WORD | 16#A9BC | |
| 2 | VAR | wInputMask | WORD | 16#000C | 2#0000000000001100 i.e. bits 2 and 3 filtered |
| 3 | VAR | wOutputData | WORD | 0 | |
| 4 | VAR | iFilterTime | INT | 100 | 0.1 seconds |

### POU body

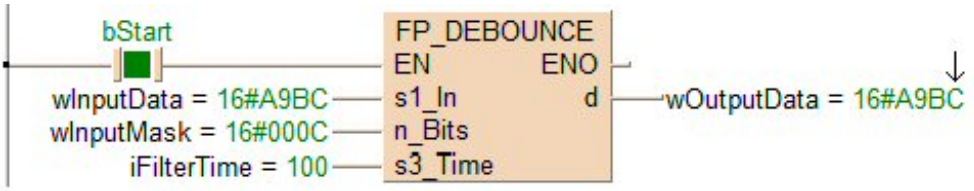
If **bStart** is TRUE, the bits of the data applied to **s1_In** are output as follows: The filtered bits will only be written to **wOutputData** after the filter time has elapsed. See time charts for a detailed explanation.

- The bits which are not set in the mask data applied to **n_Bits** are directly output without conditions.
- The bits which are set in the mask data applied to **n_Bits** are output after their signal has remained stable for the time specified by **s3_Time** in ms.

### LD body



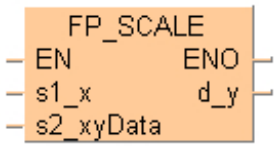

**wOutputdata** has the value 16#A9B0 for 100ms, when this time has elapsed **wOutputData** has the value 16#A9BC.



# FP_SCALE

Perform linear interpolation of discrete values

This FP instruction renders the value **y** at position **x** by performing a linear interpolation based on the neighboring reference points **Pw**$_{(xw, yw)}$ and **Pw+1**$_{(xw+1, yw+1)}$. In this example, **w** is the nearest reference point whose **x** value is smaller than the input value **s1_x**, i.e. the function connects the individual reference points in series and renders the output value **d_y** based on the input value **s1_x**.




## Parameters

Input

**s1_x**

Input value

- On 16-bit PLC types: (INT, DINT, REAL)
- On 32-bit PLC types:  (INT, DINT, UINT, UDINT, REAL, LREAL)

**s2_xyData** (user defined DUT; INT, UINT, WORD (ANY16))

Apply the first element of the user-defined DUT, i.e. the number of xy values, to this input. See description of DUT structure below.

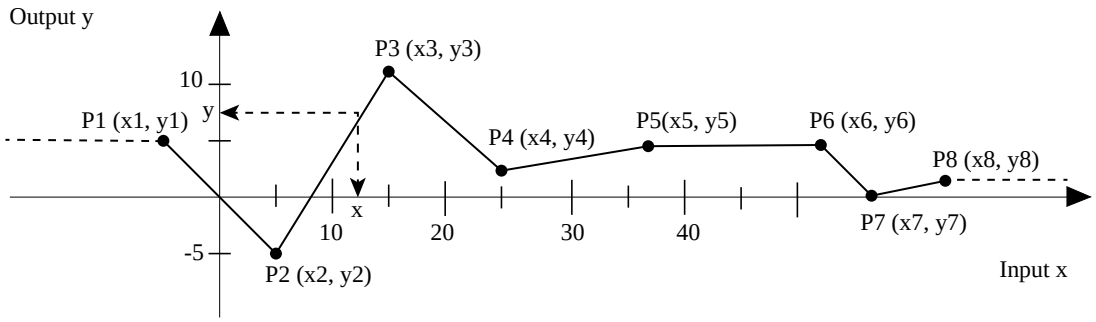
Output

**d_y**

Result

- On 16-bit PLC types: (INT, DINT, REAL)
- On 32-bit PLC types:  (INT, DINT, UINT, UDINT, REAL, LREAL)

## Remarks




Application examples:

- Linearizing measured values, e.g. with non-linear sensors
- Rendering a heater's flow temperature **y** in relation to the outside temperature **x**

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if the number of reference points is outside the range of 2–256.
- if **s2_xyData** is out of range.
- if the x values of the reference points are not in ascending order.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if the number of reference points is outside the range of 2–256.
- if **s2_xyData** is out of range.
- if the x values of the reference points are not in ascending order.

## Limitations of the output value y:

- If the input value **x** is smaller than the **x**-coordinate of the first reference point (**P1**: **x**< **x1**), the output **y** is set to the first reference point's **y**-coordinate (output **y** = **y1**, horizontal dashed line in the graph's upper left corner).
- If the input value **x** is greater than the x-coordinate of the last reference point (**P8**: **x** > **x8**), the output **y** is set to the last reference point's **y**-coordinate (output **y** = **y8**, horizontal dashed line in the graphic's upper right corner).

DUT for the xy value pairs (reference points **P1**, **P2**, ...):

The reference points (**P1**, **P2**, ...) are copied to the function via an DUT-type variable that contains the number of reference points and the **xy** value pairs (number; **x1**, **x2**, ...; **y1**, **y2**; ...).

### Structure of the user-defined DUT:

1. Element: number of reference points **z** (INT). The number of reference points (**xy** value pairs) can be set anywhere between 2–100. In the graph, eight reference points (**P1**–**P8**) are used.
2. Element: Variable of the data type ARRAY[1..z] OF INT (on 16-bit PLC types: INT, DINT, REAL; on 32-bit PLC types: (INT, DINT, UINT, UDINT, REAL)) or ARRAY[0..z-1] of INT (on 16-bit PLC types: INT, DINT, REAL; on 32-bit PLC types: (INT, DINT, UINT, UDINT, REAL)) that contains the **x** values. Here **z** represents the place marker for the number of reference points (see entry 1).
3. Element: Variable of the data type ARRAY[1..z] OF INT (on 16-bit PLC types: INT, DINT, REAL; on 32-bit PLC types: (INT, DINT, UINT, UDINT, REAL)) or ARRAY[0..z-1] of INT (on 16-bit PLC types: INT, DINT, REAL; on 32-bit PLC types: (INT, DINT, UINT, UDINT, REAL)) that contains the **y** values. Here **z** represents the place marker for the number of reference points (see entry 1).

Note

FP_SCALE supports the following data types:

- On 16-bit PLC types (FP-Sigma, FP-X): INT, DINT, REAL
- On 32-bit PLC types (FP7): INT, DINT, REAL, UINT, UDINT

## Example

### DUT

With a Data Unit Type (DUT) you can define a data unit type that is composed of other data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

xy_data_dut | FP_SCALE_LD

| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | iNumberOfValues | INT | 10 | |
| 1 | aiX | ARRAY [0..9] OF INT | [10(0)] | |
| 2 | aiY | ARRAY [0..9] OF INT | [10(0)] | |

### POU header

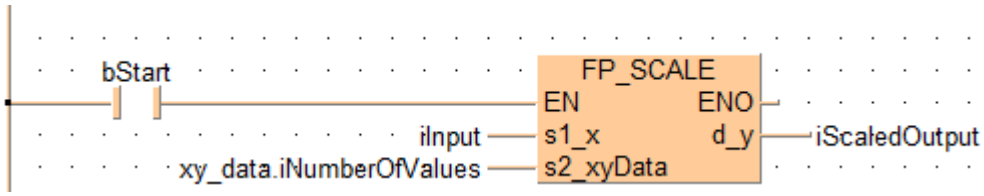
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | iInput | INT | 50 |
| 2 | VAR | xy_data | xy_data_dut | |
| 3 | VAR | iScaledOutput | INT | 0 |

## POU body

When the variable **bStart** is set to TRUE, the function is carried out.

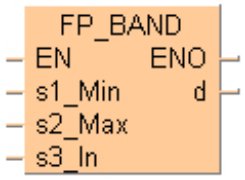
## LD body



# FP_BAND

Compare with deadband

This FP instruction compares the positive or negative input value at input **s3_In** with a deadband. The deadband's lower limit is specified at input **s1_Min** and its upper limit is specified at **s2_Max**. The result is stored in **d**.




## Parameters

Input

**s1_Min**

Input value

- On 16-bit PLC types: (INT, DINT, REAL)
- On 32-bit PLC types: (INT, DINT, UINT, UDINT, REAL, LREAL)

**s2_Max**

Input value

- On 16-bit PLC types: (INT, DINT, REAL)
- On 32-bit PLC types: (INT, DINT, UINT, UDINT, REAL, LREAL)

**s3_In**

Input value

- On 16-bit PLC types: (INT, DINT, REAL)
- On 32-bit PLC types: (INT, DINT, UINT, UDINT, REAL, LREAL)

Output

**d_y**

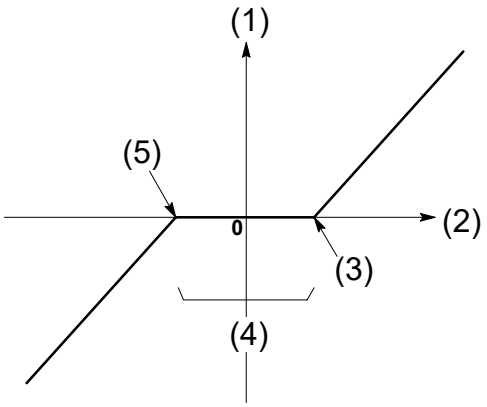
Result

- On 16-bit PLC types:  (INT, DINT, REAL)
- On 32-bit PLC types:  (INT, DINT, UINT, UDINT, REAL, LREAL)

## Remarks

The result is calculated as follows:

- For **s3_In** < **s1_Min**: **d** = **s3_In – s1_Min** (lower limit is subtracted from input value)
- For **s3_In** > **s2_Max**: **d** = **s3_In – s2_Max** (upper limit is subtracted from input value)
- For **s2_Max** ≥ **s3_In** ≥ **s1_Min**: **d** = 0




(1) Output value **d**
(2) Input value **s3_In**
(3) Upper limit of deadband **s2_Max**
(4) In this range, 0 is output
(5) Lower limit of deadband **s1_Min**

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Min** > **s2_Max**

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Min** > **s2_Max**

## Example

## POU header

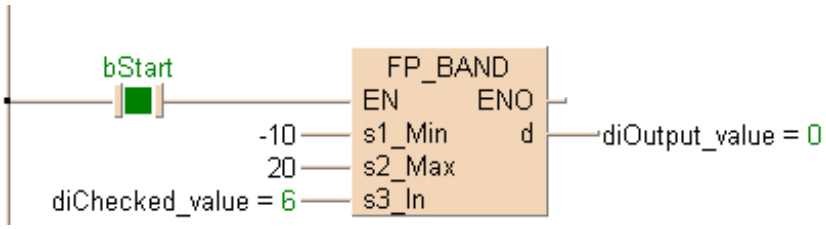
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | diChecked_value | DINT | 6 | value to check |
| 2 | VAR | diOutput_value | DINT | 0 | result after a 0->1 leading |

## POU body

When the variable **bStart** is set to TRUE, the function is carried out.

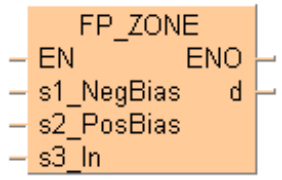
### LD body


bStart
FP_BAND
EN
ENO
-10
s1_Min
d
diOutput_value = 0
20
s2_Max
diChecked_value = 6
s3_In

# FP_ZONE

Add offset to input value

This FP instruction adds an offset value to the input value at **s3_In**. If the input value is negative, the offset specified at **s1_NegBias** is added. If the input value is positive, the offset specified at **s2_PosBias** is added. The result is stored in **d**.




## Parameters

Input

**s1_NegBias** (INT, DINT, UINT, UDINT, REAL, LREAL)

Negative bias value

**s2_PosBias** (INT, DINT, UINT, UDINT, REAL, LREAL)

Positive bias value

**s3_In** (INT, DINT, UINT, UDINT, REAL, LREAL)

Input value

Output

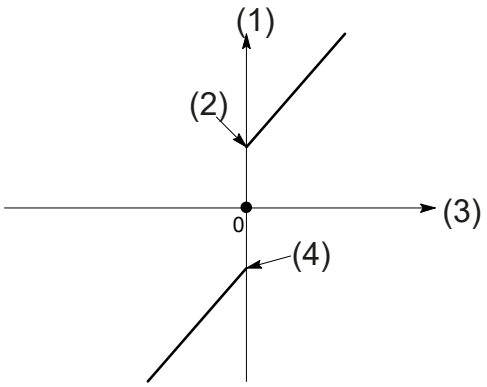
**d** (INT, DINT, UINT, UDINT, REAL, LREAL)

Result

## Remarks

The result is calculated as follows:

- For **s3_In** < 0: **d**=**s3_In** +**s1_NegBias** (negative bias is added to input value)
- **s3_In** = 0: **d**= 0
- For **s3_In** > 0: **d**=**s3_In** +**s2_PosBias** (positive bias is added to input value)




(1) Output value **d**
(2) Positive bias value **s2_PosBias**
(3) Input value **s3_In**
(4) Negative bias value **s1_NegBias**

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Min** > **s2_Max**

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the area specified using the index modifier exceeds the limit.
- if **s1_Min** > **s2_Max**

## Example

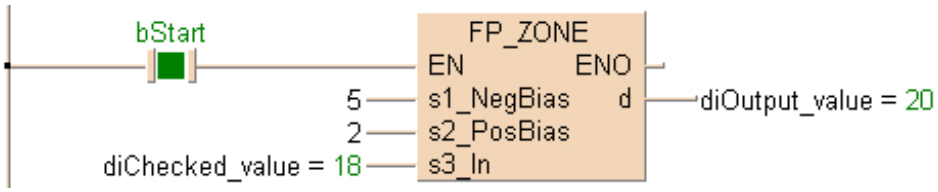
## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |
| 1 | VAR | diChecked_value | DINT | 18 | value to check |
| 2 | VAR | diOutput_value | DINT | 0 | result after a 0->1 leading |

When the variable **bStart** is set to TRUE, the function is carried out.

### LD body


bStart
FP_ZONE
EN
ENO
5
s1_NegBias
d
diOutput_value = 20
2
s2_PosBias
diChecked_value = 18
s3_In

## 29.10 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

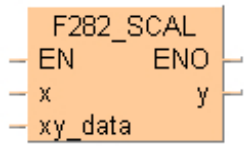
Advantages of IEC instructions

Advantages of FP instructions

# F282_SCAL

Linear interpolation of discrete INT values

The function renders the value **y** at position **x** by performing a linear interpolation based on the neighboring reference points **Pw**$_{(xw, yw)}$ and **Pw+1**$_{(xw+1, yw+1)}$. In this example, **w** is the nearest reference point whose **x** value is smaller than the input value **x**, i.e. the function connects the individual reference points in series and renders the output value **y** based on the input value **x**.




## Parameters

Input

**EN** (BOOL)

Activation of the function (when **EN** = TRUE, the function is executed during each PLC cycle)

**x** (INT)

Input value **x**

**xy_data** (user-defined DUT)

Apply the first element of the user-defined DUT, i.e. the number of xy values, to this input. See description of DUT structure below.

Output

**y** (INT)

Output value **y**

**ENO** (BOOL)

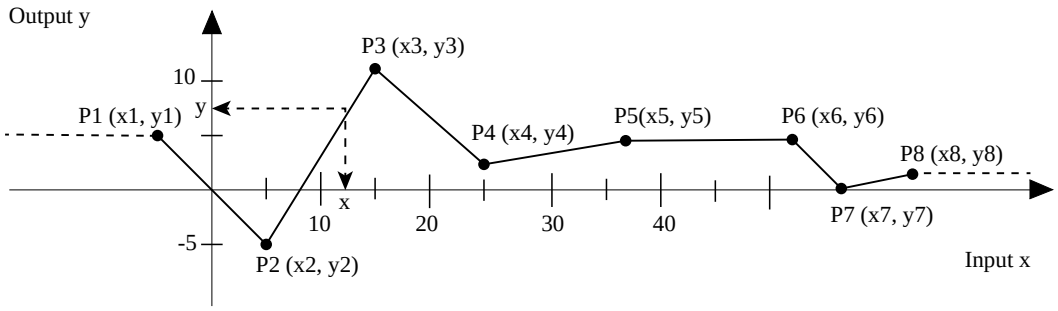
**ENO** is set to TRUE as soon the function is executed. Helpful when cascading function blocks with **EN** functions.

## Remarks

Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_SCALE** (page 1782)

The function can be used for:

- linearizing measured values, e.g. with non-linear sensors
- rendering a heater's flow temperature **y** in relation to the outside temperature **x**
- etc.




## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

if the number of reference points is not between 2 ... 100, or the **x** values are not in ascending order (**x1** < **x2** < **x3** < ...).

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

if the number of reference points is not between 2 ... 100, or the **x** values are not in ascending order (**x1** < **x2** < **x3** < ...).

## Limitations of the output value y:

If the input value **x** is smaller than the **x**-coordinate of the first reference point (P1: **x** < **x1**), the output **y** is set to the first reference point's **y**-coordinate (output **y** = **y1**, horizontal dashed line in the graph's upper left corner).

If the input value **x** is greater than the x-coordinate of the last reference point (P8: **x** > **x8**), the output **y** is set to the last reference point's **y**-coordinate (output **y** = **y8**, horizontal dashed line in the graphic's upper right corner).

DUT for the **xy** value pairs (reference points P1, P2, ...):

The reference points (P1, P2, ...) are copied to the function via an DUT-type variable that contains the number of reference points and the **xy** value pairs (number; **x1**, **x2**, ...; **y1**, **y2**; ...).

## Structure of the user-defined DUT:

1. Element: number of reference points **z** (INT). The number of reference points (**xy** value pairs) can be set anywhere between 2–100. In the graph, eight reference points (**P1**–**P8**) are used.
2. Element: **x** values (ARRAY[1..z] OF INT or ARRAY[0..z-1] of INT)
3. Element: **y** values (ARRAY[1..z] OF INT or ARRAY[0..z-1] of INT)

Important information:

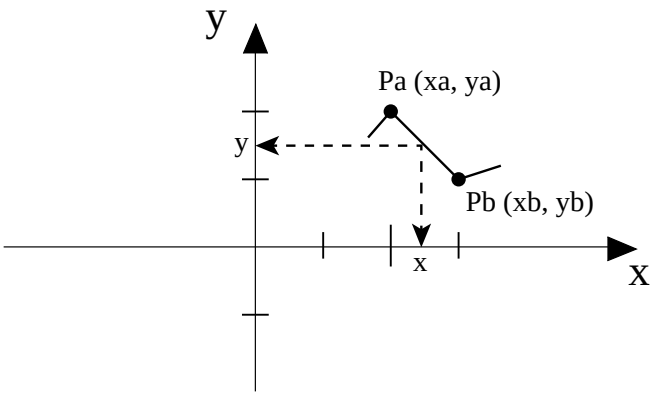
**x** values

The **x** values have to be entered in ascending order (**x1** < **x2** < **x3** < ...). If the **x** values are the same (e.g. **x2** = **x3** = **x4**) the reference points P2(**x2**,**y2**) and P3(**x3**,**y3**) are ignored.

### Overflow of the function:

In order to avoid an overflow in the calculation, neighboring reference points must fulfill the following conditions:

|ya - yb| < 32767
|x - xb| < 32767
|(ya - yb)*(x - xb)| < 32767
|xa - xb| < 32767




### Accuracy of the calculation:

This function can only process whole numbers. Numbers that follow the decimal point are cut out when calculating the value **y**. For example, if at the position **x**, **y** = 511,13, the function returns the value 511.

### Example

#### DUT

In the DUT Pool, the number of reference points and the xy value pairs are declared.

Interpolation_8_F282 [DUT]

| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | referencepoints | INT | 8 | eight reference points were |
| 1 | X_values | ARRAY [1..8] OF INT | [8(0)] | Field 1..8 -> contains 8 x-values |
| 2 | Y_values | ARRAY [1..8] OF INT | [8(0)] | Field 1..8 -> contains 8 y-values |

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

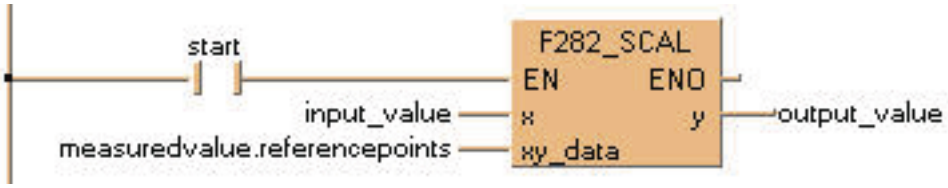
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | avtivates the function |
| 1 | VAR | input_value | INT | 0 | input_value x |
| 2 | VAR | measured_value | Interpolation_8 | X_values := [-5,5,15,20,30,42,45,50],Y_values := [5,-5,10,2,2(5),0,2] | number of reference points |
| 3 | VAR | output_value | INT | 0 | output_value y |

Here the input variable **measured_value** was declared, corresponding to the type of the DUT defined above. Assigning the x values and y values was done in the POU header. However, you can change the x values and y values in the body by assigning a value to the variable, e.g. **Measuredvalues.X_Values[1]** for x.

## POU body

When the variable **start** is set to TRUE, the function is carried out. For the input value at position x, the output value y is calculated via linear interpolation of the neighboring reference points stored in the variable **measured_value**.

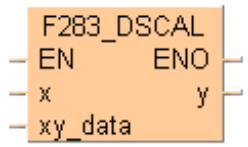
## LD body



# F283_DSCAL

Linear interpolation of discrete DINT values

The function renders the value **y** at position **x** by performing a linear interpolation based on the neighboring reference points **Pw**$_{(xw, yw)}$ and **Pw+1**$_{(xw+1, yw+1)}$. In this example, **w** is the nearest reference point whose **x** value is smaller than the input value **x**, i.e. the function connects the individual reference points in series and renders the output value **y** based on the input value **x**.




## Parameters

Input

**EN** (BOOL)

Activation of the function (when **EN** = TRUE, the function is executed during each PLC cycle)

**x** (DINT)

Input value **x**

**xy_data** (user-defined DUT)

Apply the first element of the user-defined DUT, i.e. the number of xy values, to this input. See description of DUT structure below.

Output

**ENO** (BOOL)

**ENO** is set to TRUE as soon the function is executed. Helpful when cascading function blocks with **EN** functions.

**y** (DINT)

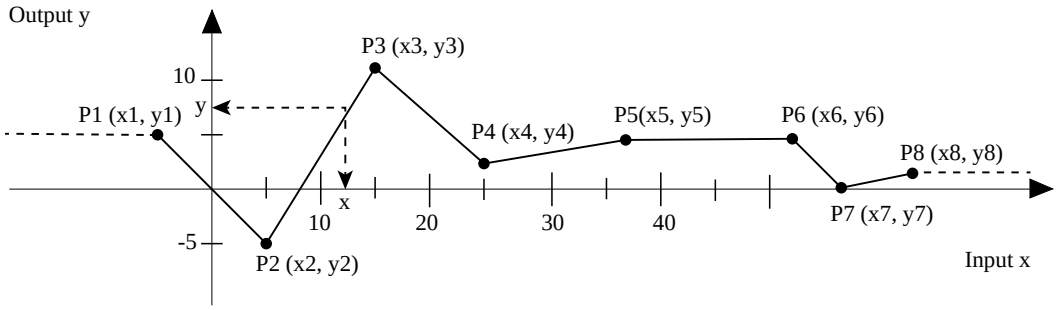
Output value **y**

## Remarks

Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_SCALE** (page 1782)

The function can be used for:

- linearizing measured values, e.g. with non-linear sensors
- rendering a heater's flow temperature **y** in relation to the outside temperature **x**
- etc.




## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if the number of reference points is not between 2 ... 100, or the **x** values are not in ascending order (**x1** < **x2** < **x3** < ...).

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if the number of reference points is not between 2 ... 100, or the **x** values are not in ascending order (**x1** < **x2** < **x3** < ...).

## Limitations of the output value y:

If the input value **x** is smaller than the x-coordinate of the first reference point (P1: **x** < **x1**), the output **y** is set to the first reference point's **y**-coordinate (output **y** = **y1**, horizontal dashed line in the graph's upper left corner).

If the input value **x** is greater than the x-coordinate of the last reference point (P8: **x** > **x8**), the output **y** is set to the last reference point's **y**-coordinate (output **y** = **y8**, horizontal dashed line in the graphic's upper right corner).

DUT for the **xy** value pairs (reference points P1, P2, ...):

The reference points (P1, P2, ...) are copied to the function via a DUT-type variable that contains the number of reference points and the **xy** value pairs (number; **x1**, **x2**, ...; **y1**, **y2**; ...).

### Structure of the user-defined DUT:

1. Element: number of reference points **z** (INT). The number of reference points (**xy** value pairs) can be set anywhere between 2–100. In the graph, eight reference points (**P1**–**P8**) are used.
2. Element: **x** values (ARRAY[1..z] OF DINT or ARRAY[0..z-1] of DINT)
3. Element: **y** values (ARRAY[1..z] OF DINT or ARRAY[0..z-1] of DINT)

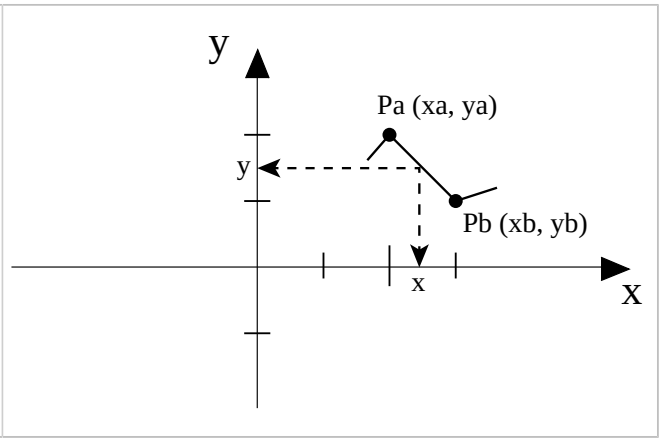
Note

The **x** values have to be entered in an ascending order (**x1** < **x2** < **x3** < ...). If the **x** values are the same (e.g. **x2** = **x3** = **x4**) the reference points P2(**x2**,**y2**) and P3(**x3**,**y3**) are ignored.

### Overflow of the function:

In order to avoid an overflow in the calculation, neighboring reference points must fulfill the following conditions:

|ya - yb| < 2147483647
|x - xb| < 2147483647
|(ya - yb)*(x - xb)| < 2147483647
|xa - xb| < 2147483647




### Accuracy of the calculation:

This function can only process whole numbers. Numbers that follow the decimal point are cut out when calculating the value **y**. For example, if at the position **x**, **y** = 511,13, the function returns the value 511.

### Example

#### DUT

In the DUT Pool, the number of reference points and the xy value pairs are declared.

Interpolation_8_F283 [DUT]

| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | referencepoints | INT | 8 | eight reference points were |
| 1 | X_values | ARRAY [1..8] OF DINT | [8(0)] | Field 1..8 -> contains 8 x-values |
| 2 | Y_values | ARRAY [1..8] OF DINT | [8(0)] | Field 1..8 -> contains 8 y-values |

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

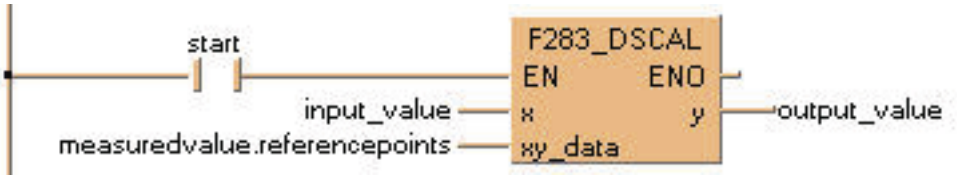
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | avtivates the function |
| 1 | VAR | input_value | DINT | 0 | input_value x |
| 2 | VAR | measured_value | Interpolation_8 | X_values := [-5,5,15,20,30,42,45,50],Y_values := [5,-5,10,2,2(5),0,2] | number of reference points |
| 3 | VAR | output_value | DINT | 0 | output_value y |

Here the input variable **measured_value** was declared, corresponding to the type of the DUT defined above. Assigning the x values and y values was done in the POU header. However, you can change the x values and y values in the body by assigning a value to the variable, e.g. **Measuredvalues.Y_Values[3]** for y3.

## POU body

When the variable **start** is set to TRUE, the function is carried out. For the **input_value** at position x, the output value y is calculated via linear interpolation between the neighboring reference points stored in the variable **measured_value**.

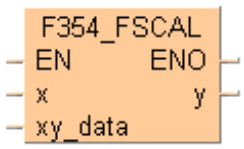
## LD body



# F354_FSCAL

Linear interpolation of discrete REAL values

This function performs scaling (linearization) of a real number data table and renders the output (**Y**) for an input value (**X**).




## Parameters

Input

**x** (REAL)

Input value **X**

**xy_data** (user-defined DUT)

Apply the first element of the user-defined DUT, i.e. the number of xy values, to this input. See description of DUT structure below.

Output

**y** (REAL)

Output value **Y**

## Remarks

Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_SCALE** (page 1782)

For a detailed description, refer to the instructions: **F282_SCAL** (page 1793) and**F283_DSCAL** (page 1797).

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if the specified address using the index modifier exceeds a limit.
- if a non-real number value is input into '**x**'.
- if the number of values (first element of the DUT) < 2 or > 99.
- if a non- real number value is specified to be the real numerical value (**xt**, **yt**) specified in '**xy**_data'.

- if the linear table of '**xy**_data' is not registered in ascending order of the **x**-sequence.
- if the linear table of '**xy**_data' exceeds the area.
- if an overflow (operation is unable) occurs during the scaling operation.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if the specified address using the index modifier exceeds a limit.
- if a non-real number value is input into '**x**'.
- if the number of values (first element of the DUT) < 2 or > 99.
- if a non- real number value is specified to be the real numerical value (**xt**, **yt**) specified in '**xy**_data'.
- if the linear table of '**xy**_data' is not registered in ascending order of the **x**-sequence.
- if the linear table of '**xy**_data' exceeds the area.
- if an overflow (operation is unable) occurs during the scaling operation.

## Example

### DUT

The Data Unit Type is created in the DUT Pool.

XY_DUT [DUT]

| | Identifier | Type | Initial |
|---|---|---|---|
| 0 | Number | INT | 6 |
| 1 | X_values | ARRAY [1..6] OF REAL | [1.0,1.5,2.0,5.0,5.5,6.0] |
| 2 | Y_values | ARRAY [1..6] OF REAL | [1.0,1.3,2.5,10.0,11.0,9.0] |

### POU header

All input and output variables used for programming this function have been declared in the POU header.

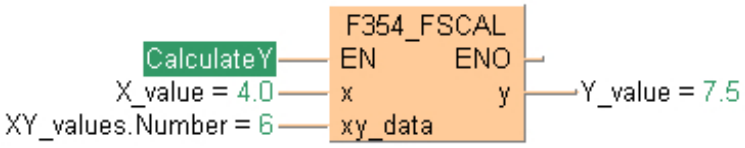
The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | CalculateY | BOOL | FALSE |
| 1 | VAR | X_value | REAL | 4.0 |
| 2 | VAR | Y_value | REAL | 0.0 |
| 3 | VAR | XY_values | XY_DUT | |

### POU body

When the variable **CalculateY** is set to TRUE, the function is carried out.

### LD body


F354_FSCAL
CalculateY
EN
ENO
X_value = 4.0
x
y
Y_value = 7.5
XY_values.Number = 6
xy_data

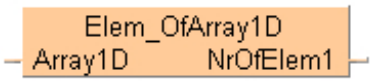
# 30 Size information instructions

# Elem_OfArray1D

Number of elements in an array

This function yields the number of elements in an array.

The total number of elements of the array variable at input **Array1D** is yielded at output **NrOfElem1**. An array variable of any dimension preferred can be transferred. In any case, the total number of elements of all dimensions is returned. Hence, functions and function blocks can be written that process arrays of various lengths (also in conjunction with the functions (**GetPointer**, **AreaOffs_ToVar**,**Var_ToAreaOffs**).




### Parameters

Input

**Array1D** (ANY_IN_UNITS_OF_WORDS)

Input variable (array dimension 1,2 or 3) whose number of elements is to be determined

Output

**NrOfElem1** (INT, DINT, UINT, UDINT)

Yields number of elements in the input variable

## Example

### POU header

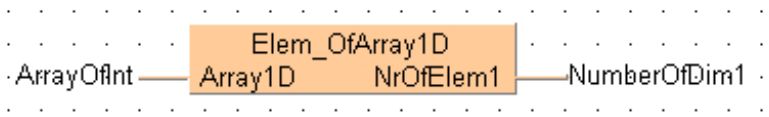
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | ArrayOfInt | ARRAY [2..15] OF INT | [14(0)] | one-dimensional data field |
| 1 | VAR | NumberOfDi m 1 | INT | 0 | result:14 |

### POU body

In this case the function **Elem_OfArray1D** is executed in each CPU cycle (no EN input). It determines the number of elements of the variable **ArrayOfInt**. The result here is: **NumberOfDim1** = 14.

### LD body

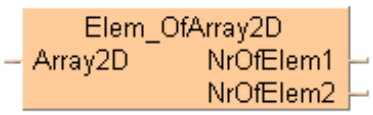



Further example projects (directory "Samples" of FPWIN Pro installation):Calculate the sum of all array elements

# Elem_OfArray2D

Number of elements in an array

This function yields the number of elements of the 1st and 2nd dimension of an array.




## Parameters

Input

**Array2D** (ANY_IN_UNITS_OF_WORDS)

Input variable (array dimension 1,2 or 3) whose number of elements is to be determined

Output

**NrOfElem1** (INT, DINT, UINT, UDINT)

Yields number of elements in the first dimension

**NrOfElem2** (INT, DINT, UINT, UDINT)

Yields number of elements in the second dimension

## Remarks

If input **Array2D** is an one dimensional array:

- The number of elements in the array at input **Array2D** is yielded at output **NrOfElem1**.
- Value 1 is returned at output **NrOfElem2**.

If input **Array2D** is a two dimensional array:

- From the array at input **Array2D**, the number of elements in the first dimension is yielded at output **NrOfElem1** and the number of elements in the second dimension at output **NrOfElem2**.

If input **Array2D** is a three dimensional array:

- From the array at input **Array2D**, the number of elements in the first dimension is yielded at output **NrOfElem1** and the product of the array's second and third dimensions at output **NrOfElem2**.

Hence, functions and function blocks can be written that process arrays of various lengths (also in conjunction with the functions (**GetPointer**, **AreaOffs_ToVar**,**Var_ToAreaOffs**).

Note

The product **NrOfElem1*NrOfElem2** always equals the total number of the elements in the array.

## Example

### POU header

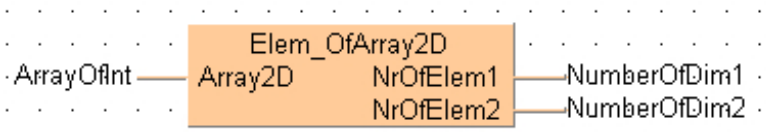
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | ArrayOfInt | ARRAY [2..15,3..20] OF INT | [252(0)] | two-dimensional data field |
| 1 | VAR | NumberOfDi m 1 | INT | 0 | result: 14 |
| 2 | VAR | NumberOfDi m 2 | INT | 0 | result: 18 (14*18=252) |

### POU body

In this case the function **Elem_OfArray2D** is executed in each CPU cycle (no EN input). It determines the number of elements of the variable **ArrayOfInt**. The result here is: **NumberOfDim1** = 14 and **NumberOfDim2** = 18.

### LD body

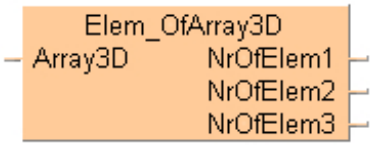



Further example projects (directory "Samples" of FPWIN Pro installation):Calculate the product of two-dimensional arrays

# Elem_OfArray3D

Number of elements in an array

This function yields the number of elements of the 1st, 2nd and 3rd dimension of an array.




## Parameters

Input

**Array3D** (ANY_IN_UNITS_OF_WORDS)

Input variable (array dimension 1,2 or 3) whose number of elements is to be determined

Output

**NrOfElem1** (INT, DINT, UINT, UDINT)

Yields number of elements in the first dimension

**NrOfElem2** (INT, DINT, UINT, UDINT)

Yields number of elements in the second dimension

**NrOfElem3** (INT, DINT, UINT, UDINT)

Yields number of elements in the third dimension

## Remarks

If input **Array3D** is an one dimensional array:

- The number of elements in the array at input **Array3D** is yielded at output **NrOfElem1**.
- Value 1 is returned at output **NrOfElem2**.
- Value 1 is returned at output **NrOfElem3**.

If input **Array3D** is a two dimensional array:

- From the array at input **Array3D**, the number of elements in the first dimension is yielded at output **NrOfElem1** and the number of elements in the second dimension at output **NrOfElem2**.
- Value 1 is returned at output **NrOfElem3**.

If input **Array3D** is a three dimensional array:

- At outputs **NrOfElem1**, **NrOfElem2** and **NrOfElem3** the number of elements in the first, second, and third dimensions of the array at input **Array3D** are yielded.

Hence, functions and function blocks can be written that process arrays of various lengths (also in conjunction with the functions (**GetPointer**, **AreaOffs_ToVar**,**Var_ToAreaOffs**).

Note

The product **NrOfElem1** * **NrOfElem2** * **NrOfElem3** always equals the total number of the elements in the array.

## Example

### POU header

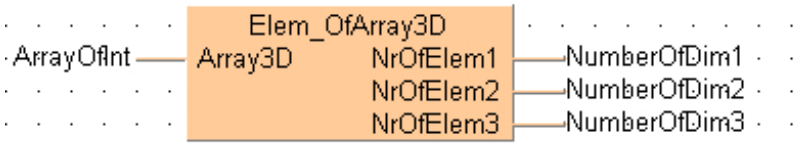
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | ArrayOfInt | ARRAY [2..15,3..20,4..25] OF INT | [5544(0)] | three-dimensional data field |
| 1 | VAR | NumberOfDi m 1 | INT | 0 | result: 14 |
| 2 | VAR | NumberOfDi m 2 | INT | 0 | result: 18 |
| 3 | VAR | NumberOfDi m 3 | INT | 0 | result: 22 (14*18*22=5544) |

### POU body

In this case the function **Elem_OfArray3D** is executed in each CPU cycle (no EN input). It determines the number of elements of the variable **ArrayOfInt**. The result here is: **NumberOfDim1** = 14, **NumberOfDim2** = 18 and **NumberOfDim3** = 22.

### LD body

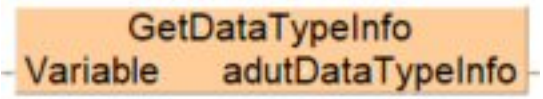



Further example projects (directory "Samples" of FPWIN Pro installation):Calculate the trace of a three-dimensional array (FB)

# GetDataTypeInfo

Read the description of a structured variable

This instruction gets the variable description of a structured variable (DUT, ARRAY) at the input **Variable** and writes the description into the output variable **adutDataTypeInfo**.




Input

**Variable** (BOOL, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, DATE, TOD, DT, STRING)

Start address of the structured variable

The variable description will be written to an ARRAY OF **DATA_TYPE_INFO_DUT**, the size of which must correspond to the number of DUT elements of the variable specified at the input **Variable**. Each element of the variable description describes an element of the DUT.

Output

**adutDataTypeInfo** (ANY ARRAY OF **DATA_TYPE_INFO_DUT**)

Start address of the variable description array

## Remarks

- Valid types for the input **Variable** are structured data types with member variables of any simple type (except BOOL), array of those simple types and data unit types consisting of those simple types
- The variable **adutDataTypeInfo** must be an array of **DATA_TYPE_INFO_DUT**
- The size of the array must be equal to or greater than the number of elements in the process data.

## Example

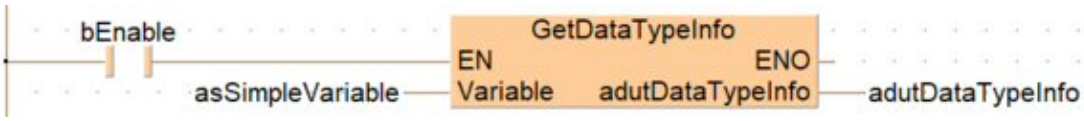
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | asSimpleVariable | ARRAY [0..2] OF STRING[32] | [3('')] |
| 3 | VAR | adutDataTypeInfo | ARRAY [0..0] OF DATA_TYPE_INFO_DUT | |

### LD body

When the variable **bEnable** is set to TRUE, the function is executed.




## Example

DUT
In this example, the variable **dutVariable** is a DUT of the type **Example_2_DUT** with the following structure:

Example_2_DUT

| | Identifier | Type | Initial |
|---|---|---|---|
| 1 | uiValue | UINT | 0 |
| 2 | arValues | ARRAY [0..2] OF REAL | [3(0)] |
| 3 | asValues | ARRAY [0..2] OF STRING[32] | [3('')] |

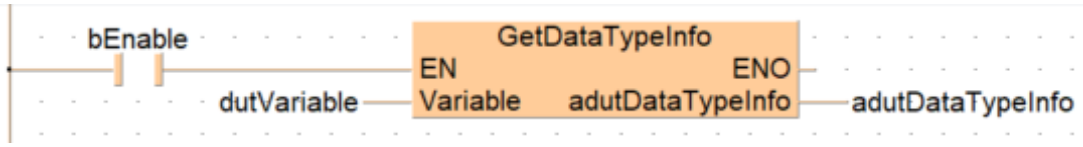
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bEnable | BOOL | FALSE |
| 2 | VAR | dutVariable | Example_2_DUT | |
| 3 | VAR | adutDataTypeInfo | ARRAY [0..2] OF DATA_TYPE_INFO_DUT | |

### LD body

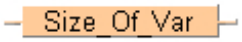
When the variable **bEnable** is set to TRUE, the function is executed.



# Size_Of_Var

Returns the size of a variable

This function yields the size of a variable in words (with Enable).




## Parameters

Input

**Var** (ANY_IN_UNITS_OF_WORDS)

Input variable whose size is to be determined

Output

**Size** (INT, DINT, UINT, UDINT)

Yields the size of the input variable in Word units (16-bit)

## Remarks

The size of the variable at input **Var** is yielded at output **Size** in Word units (16-bit).

# 31 Special instructions

## 31.1 FP instructions

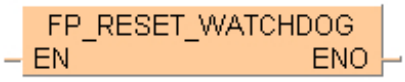
Tip

Advantages of FP instructions

# FP_RESET_WATCHDOG

Reset the watchdog timer

This FP instruction resets the watchdog timer to zero if the trigger **EN** is TRUE.




## Example

### POU header

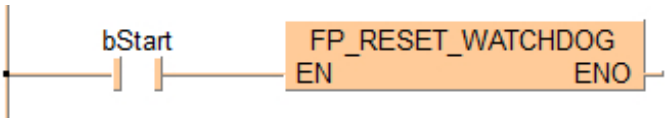
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the instruction |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

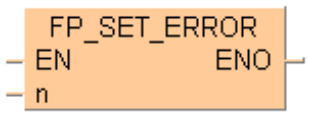
### LD body



# FP_SET_ERROR

Set/reset self-diagnostic error

The error no. specified by **n** is copied into the system variable **sys_iSelfDiagnosticErrorCode** that reads the corresponding special data register.




## Parameters

Input

**n** (INT)

must be a constant

- Self-diagnostic error code number, range: 0 and 100 to 299
- For FP7: range 0! and 1000–2999

## Remarks

The contents of the system variable **sys_iSelfDiagnosticErrorCode** and the error no. can be read and checked using Control FPWIN Pro “Monitor” > “Special flags and registers” > “Basic error messages”.

Error number areas:

- When **n**=0, all error numbers greater than 43 are cleared and the error LED turns off.
- When **n**=100–199, the operation is halted.
- When **n**=200–299, the operation is continued.

For FP7:

- When **n** = 0!, no error
- When **n** < 1000, operation error occurs
- When 1000 < **n** < 1999, PLC stops
- When 2000 < **n** < 2999, error occurs but PLC continues

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if **n** exceeds the limit.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if **n** exceeds the limit.

## Example

### POU header

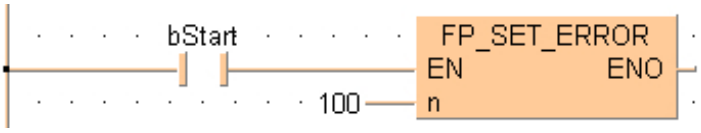
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE | activates the function |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

### LD body



## 31.2 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

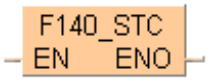
Advantages of IEC instructions

Advantages of FP instructions

# F140_STC

Carry-flag set

The system variable **sys_bIsCarry** (carry-flag) goes ON if the trigger **EN** is in the ON-state. This instruction can be used to control data using carry-flag R9009 (e.g.**F122_RCR** (page ) and **F123_RCL** (page ) instructions).




## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function; result after a leading edge from start: carry-flag (R9009) will be set ON |
| 1 | VAR | | | | |

## POU body

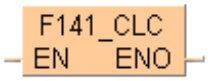
When the variable **start** is set to TRUE, the function is carried out.

## LD body

# F141_CLC

Carry-flag reset

Special internal relay R9009 (carry-flag) goes OFF if the trigger **EN** is in the ON-state. This instruction can be used to control data using carry-flag R9009 (e.g.**F122_RCR** (page ) and **F123_RCL** instructions).




## Example

### POU header

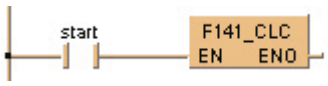
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function; result after a leading edge from start: carry-flag (R9009) will be set OFF |
| 1 | VAR | | | | |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

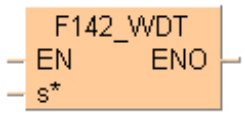
### LD body



# F142_WDT

Watchdog timer update

The scan ‘check watchdog timer’ is preset with the constant specified by **s** if the trigger **EN** is in the ON-state. The value specified by **s*** is 1 to 255 and the preset time becomes 2.5 ms * **s*** (637.5 ms).




## Parameters

Input

**s*** (INT)

specifies watchdog timer value

## Remarks

- Instead of using this F instruction, we recommend using the corresponding FP7 instruction: **FP_RESET_WATCHDOG** (page 1816)
- The scan ‘check watchdog timer’ is automatically set at the start of a scan with the value of the system register (No. 30). To monitor the transit of a processing block, set the watchdog timer with this instruction immediately before transition and set again immediately after that.

## Example

### POU header

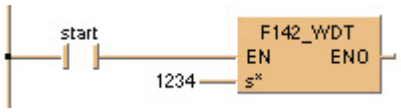
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |

### POU body

When the variable **start** is set to TRUE, the function is carried out.

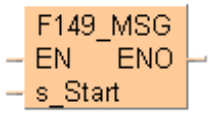
### LD body



# F149_MSG

Message display

This instruction is used for displaying the message on the FP Programmer II screen. After executing the **F149_MSG** instruction, you can see the message specified by **s_Start** on the FP Programmer II screen.




## Parameters

Input

**s_Start** (STRING[12])

Message to be displayed

## Remarks

When the **F149_MSG** instruction is executed, the message-flag R9026 is set and the message specified by **s_Start** is set in special data registers DT9030 to DT9035 (DT90030 to DT90035 for FP0 T32CP, FP2/2SH, FP10/10S/10SH). Once the message is set in special data registers, the message cannot be changed even if the **F149_MSG** instruction is executed again. You can clear the message with the FP Programmer II.

## Example

## POU header

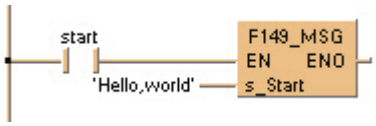
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | start | BOOL | FALSE | activates the function |

## POU body

When the variable **start** is set to TRUE, the function is carried out.

### LD body

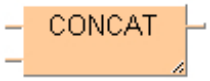


# 32 String instructions

# CONCAT

Concatenate (attach) a string

**CONCAT** concatenates (attaches) the second and the following input strings (IN1 + IN2 + ...) to the first input string and writes the resulting string into the output variable.




## Parameters

Input

**Unnamed input** (STRING)

Input string

**Unnamed input** (STRING)

String that will be attached to the beginning string
The number of input contacts is expandable, see also modifying elements.

Output

**Unnamed output** (STRING)

Resulting string

## Remarks

- If this instruction is used with UTF-8 strings, please refer to the notes concerning UTF-8 strings under the data type STRING.
- If the output string is longer than the length defined for the output variable in the field “Type”, only as many characters are copied as the output variable can hold.The system variable **sys_bIsCarry** is set.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if a string applied at the input or output is an invalid string

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if a string applied at the input or output is an invalid string

**sys_blsCarry** (turns to TRUE for one scan)

if the output string is longer than the length defined for the output variable in the field "Type"

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

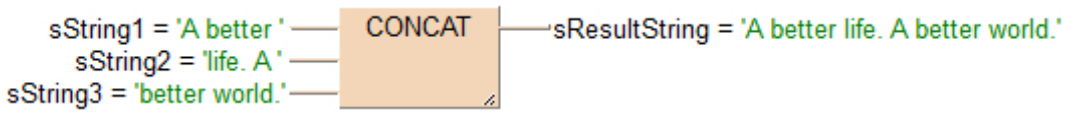
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | sString1 | STRING[32] | 'A better ' | sample string 1 |
| 1 | VAR | sString2 | STRING[32] | 'life. A ' | sample string 2 |
| 2 | VAR | sString3 | STRING[32] | 'better world.' | sample string 3 |
| 3 | VAR | sResultString | STRING[32] | '' | result: 'A better life. A better world.' |

In this example the input variables (**sString1**, **sString2** and **sString3**) have been declared. However, you may enter the strings directly into the function. The strings have to be put in inverted commas, both in the POU header and in the function.

### POU body

**sString3** is attached to **sString2** and this string is attached to **sString1**. The resulting string is written into **sResultString**.

### LD body



# DELETE

Delete characters from a string

**DELETE** deletes **L** characters in the string **IN** starting at position **P**, where 1 denotes the first character of the string. The result is written into the output variable.


## Parameters

Input

**IN** (STRING)

Input string

**L** (INT)

Number of input string's characters that are deleted

**P** (INT)

Start position of deletion, where 1 denotes the first character of the string

Output

**Unnamed output** (STRING)

Resulting string

## Remarks

- If this instruction is used with UTF-8 strings, please refer to the notes concerning UTF-8 strings under the data type STRING.
- If the output string is longer than the length defined for the output variable in the field “Type”, only as many characters are copied as the output variable can hold.The system variable **sys_bIsCarry** is set.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if a string applied at the input or output is an invalid string

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if a string applied at the input or output is an invalid string

**sys_bIsCarry** (turns to TRUE for one scan)

if the output string is longer than the length defined for the output variable in the field "Type"

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

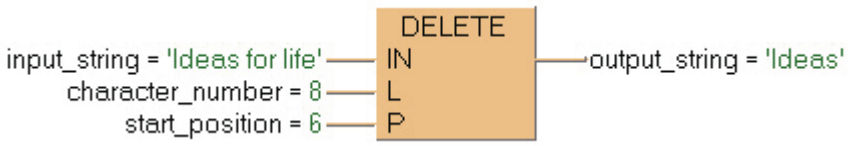
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_string | STRING[15] | 'Ideas for life' | sample string |
| 1 | VAR | character_number | INT | 8 | characters to be deleted |
| 2 | VAR | start_position | INT | 6 | position to start deleting |
| 3 | VAR | output_string | STRING[5] | '' | result: here 'Ideas' |

In this example the input variables (**input_string**, **character_number** and **start_position**) have been declared. Instead, you may enter the string ('Ideas for life'), the number of characters to be deleted and the start position directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

### POU body

Starting from **start_position**(6), **character_number** (8) is deleted from **input_string** ('Ideas for life'). The resulting string ('Ideas') is written into **output_string**.

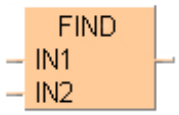
### LD body



# FIND

Find string's position

**FIND** returns the position at which the second input string first occurs in the first input string. The result is written into the output variable. If the second input string does not occur in the first input string, the value ZERO is returned.




## Parameters

Input

**IN1** (STRING)

Input string

**IN2** (STRING)

Case-sensitive string that is searched for in the input string

Output

**Unnamed output** (INT)

- if value > 0: position at which the string searched for is found, whereby 1 refers to the first character
- if value = 0: search string not found

## Remarks

- If this instruction is used with UTF-8 strings, please refer to the notes concerning UTF-8 strings under the data type STRING.
- If the strings are longer than the length defined for the input variables (**IN1** and **IN2**) in the declaration field “Type”, an error occurs (see **sys_bIsCarry** for error handling).

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if a string applied at the input or output is an invalid string
- if the input strings are longer than the length defined for the input variables in the field “Type”

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if a string applied at the input or output is an invalid string
- if the input strings are longer than the length defined for the input variables in the field “Type”

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

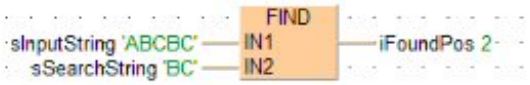
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 1 | VAR | sInputString | STRING[5] | 'ABCBC' | sample string |
| 2 | VAR | sSearchString | STRING[2] | 'BC' | searched string |
| 3 | VAR | iFoundPos | INT | 0 | 1st position found |

In this example the input variables (**sInputString** and **sSearchString**) have been declared. Instead, you may enter the strings ('ABCBC.' and 'BC') directly into the function. The strings have to be put in inverted commas, both in the POU header and in the function.

### POU body

**sSearchString** ('BC’) is searched in **sInputString** ('ABCBC’). The position of the first occurrence (2) is written into **iFoundPos**.

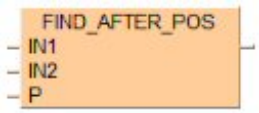
### LD body



# FIND_AFTER_POS

Find string's position

**FIND_AFTER_POS** returns the position at which the second input string **IN2** occurs in the first input string **IN1** beginning from the start position **P**. The result is written into the output variable. If the second input string does not occur in the first input string, the value ZERO is returned.




## Parameters

Input

**IN1** (STRING)

Input string

**IN2** (STRING)

Case-sensitive string that is searched for in the input string

**P** (INT)

Start position of the input string to be searched, where 1 is the first character of the string

Output

**Unnamed output** (INT)

Position at which the string searched for is found

- if value > 0: position at which the string searched for is found, whereby 1 refers to the first character
- if value = 0: search string not found

## Remarks

- If this instruction is used with UTF-8 strings, please refer to the notes concerning UTF-8 strings under the data type STRING.
- If the strings are longer than the length defined for the input variables (**IN1** and **IN2**) in the declaration field “Type”, an error occurs (see **sys_blsCarry** for error handling).

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if a string applied at the input or output is an invalid string

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if a string applied at the input or output is an invalid string

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

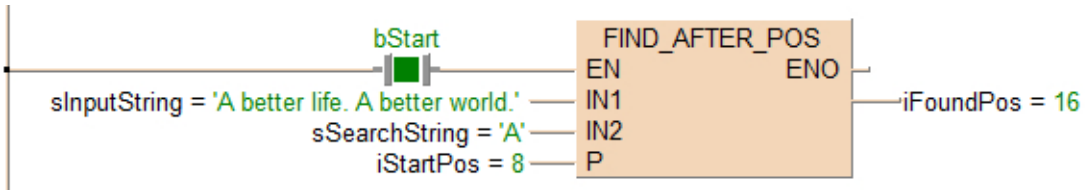
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bStart | BOOL | FALSE |
| 1 | VAR | sInputString | STRING[32] | 'A better life. A better world.' |
| 2 | VAR | sSearchString | STRING[32] | 'A' |
| 3 | VAR | iStartPos | INT | 8 |
| 4 | VAR | iFoundPos | INT | 0 |

In this example the input variables **sSearchString** and **sInputString** have been declared. Instead, you may enter the strings directly into the function. The strings have to be put in inverted commas, both in the POU header and in the function.

### POU body

**sSearchString** is searched in **sInputString** beginning from start position 8. The position of the first occurrence after position 8 is written into **iFoundPos**.

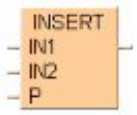
### LD body



# INSERT

Insert characters

**INSERT** inserts the string **IN2** into the string **IN1** beginning after the character position **P**, where 0 denotes the beginning of the string, 1 the position after the first string character etc. The result is written into the output variable.




## Parameters

Input

**IN1** (STRING)

input string

**IN2** (STRING)

string to be inserted into input string

**P** (INT)

position after which string **IN2** is inserted into input string **IN1**, where 0 denotes the beginning of the string, 1 the position after the first string character

Output

**Unnamed output** (STRING)

result string

## Remarks

- If this instruction is used with UTF-8 strings, please refer to the notes concerning UTF-8 strings under the data type STRING.
- If the strings are longer than the length defined for the input variables (**IN1** and **IN2**) in the declaration field “Type”, an error occurs (see **sys_bIsCarry** for error handling).

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if a string applied at the input or output is an invalid string

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if a string applied at the input or output is an invalid string

## Example

### POU header

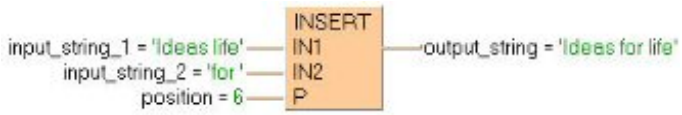
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_string1 | STRING[32] | 'ideas life' | sample string |
| 1 | VAR | input_string2 | STRING[32] | 'for' | sample string |
| 2 | VAR | position | INT | 6 | |
| 3 | VAR | output_string | STRING[32] | '' | result: here 'Ideas for life' |

### POU body

In this example the input variables **input_string1**, **input_string2** and **position** have been declared. However, you may enter the values directly at the function's input contact pins instead. The STRING values have to be put in inverted commas, both in the POU header and at the contact pins. **input_string2** ('for ') is inserted into **input_string1** ('Ideas life') after character position 6. The result ('Ideas for life') is returned at **output_value**. In the LD example, (Monitoring) icon was activated while in online mode, hence you can see the results immediately.

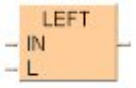
### LD body



# LEFT

Copy characters from the left

LEFT copies, starting from the left, **n** characters of the string of the first input variable to the output variable. You define the number of characters **L** to be delivered by the second input variable.




## Parameters

Input

**IN** (STRING)

input string

**L** (INT)

number of input string's characters that are copied, from the left

Output

**Unnamed output** (STRING)

copied string

## Remarks

- If this instruction is used with UTF-8 strings, please refer to the notes concerning UTF-8 strings under the data type STRING.
- If the number of characters to be delivered is greater than the input string, the complete string will be copied to the output variable **output_string**.
- If the output string is longer than the length defined for the output variable in the field “Type”, only as many characters are copied as the output variable can hold.The system variable **sys_bIsCarry** is set.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if a string applied at the input or output is an invalid string

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if a string applied at the input or output is an invalid string

**sys_blsCarry** (turns to TRUE for one scan)

if the output string is longer than the length defined for the output variable in the field "Type"

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

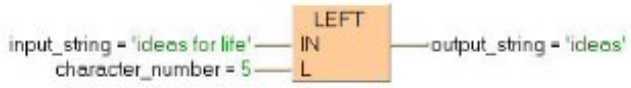
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_string | STRING[15] | 'Ideas for l... | sample string |
| 1 | VAR | output_string | STRING[5] | '' | result: here 'Ideas' |
| 2 | VAR | character_number | INT | 5 | characters to be delivered |

In this example the input variables (**input_string** and **character_number**) have been declared. Instead, you may enter the string ('Ideas for life') and the number of characters to be delivered directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

### POU body

Starting from the left, **character_number** (5) of **input_string** ('Ideas for life') is copied to **output_string** ('Ideas').

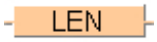
### LD body



# LEN

String length

**LEN** calculates the length of the input string and writes the result into the output variable.




## Parameters

Input

**Unnamed input** (STRING)

input data type

Output

**Unnamed output** (INT)

length of string

## Remarks

- If this instruction is used with UTF-8 strings, please refer to the notes concerning UTF-8 strings under the data type STRING.
- If the strings are longer than the length defined for the input variables (**IN1** and **IN2**) in the declaration field “Type”, an error occurs (see **sys_bIsCarry** for error handling).

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if a string applied at the input or output is an invalid string

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if a string applied at the input or output is an invalid string

# Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | input_string | STRING[12] | 'Panasonic' | sample string |
| 1 | VAR | output_value | INT | 0 | result: here 9 |

In this example the input variable (**input_string)** has been declared. Instead, you may enter the string ('Panasonic') directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

### POU body

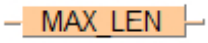
The length (9) of **input_string** ('Panasonic') is written into **output_value**.

### LD body

# MAX_LEN

Returns the maximum string length

This instruction returns the maximum string length set in the variable declaration from the input variable and writes the value into the output variable.




## Parameters

Input

**Unnamed input** (STRING)

String variable declared in the POU header

Output

**Unnamed output** (INT)

Maximum number of characters of declaration

## Remarks

- If this instruction is used with UTF-8 strings, please refer to the notes concerning UTF-8 strings under the data type STRING.

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

if a string applied at the input or output is an invalid string

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

if a string applied at the input or output is an invalid string

# Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

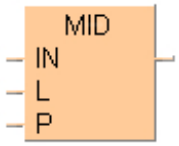
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | sTestString | STRING[128] | 'A better life. A better world' |
| 1 | VAR | iStringLength | INT | 0 |
| 2 | VAR | iMaxStringLength | INT | 0 |

**LD body**

# MID

Copy characters from a middle position

**MID** copies **L** characters of the string **IN** starting at position **P** with 1 denoting the first character of the string. The result is written into the output variable.




## Parameters

Input

**IN** (STRING)

Input string

**L** (INT)

Number of input string's characters that are copied

**P** (INT)

Start position of the input string to be copied, where 1 is the first character of the string

Output

**Unnamed output** (STRING)

Copied string

## Remarks

- If this instruction is used with UTF-8 strings, please refer to the notes concerning UTF-8 strings under the data type STRING.
- The sum of start position and number of characters to be delivered should not be greater than the input string. If you want to receive for example 5 characters of a 10-character string, starting from position 7, only the last 4 characters are delivered.
- If the output string is longer than the length defined for the output variable in the field “Type”, only as many characters are copied as the output variable can hold.The system variable **sys_blsCarry** is set.

### Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

if a string applied at the input or output is an invalid string

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

if a string applied at the input or output is an invalid string

**sys_blsCarry** (turns to TRUE for one scan)

if the output string is longer than the length defined for the output variable in the field "Type"

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

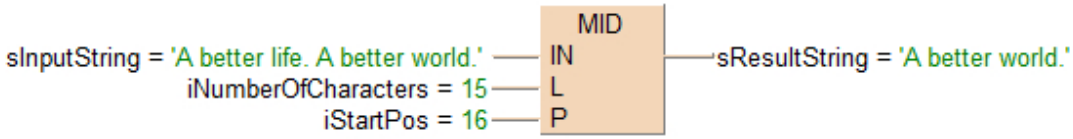
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | sInputString | STRING[32] | 'A better life. A better world.' | sample string |
| 1 | VAR | iNumberOfCharacters | INT | 15 | characters to be delivered |
| 2 | VAR | iStartPos | INT | 16 | position to start copying |
| 3 | VAR | sResultString | STRING[15] | '' | result here: 'A better world.' |

In this example the input variables (**sInputString**, **iNumberOfCharacters** and **iStartPos**) have been declared. Instead, you may enter the string, the number of characters to be delivered and the start position directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

### POU body

Starting from **iStartPos** (16), **iNumberOfCharacters** (15) of **sInputString** is copied to **sResultString**.

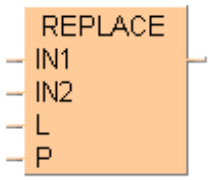
### LD body



# REPLACE

Replaces characters

**REPLACE** replaces the characters in the string **IN1** with **P** denoting the first position to be replaced and **L** denoting the number of characters to be replaced with the characters specified by **IN2**. The result is written into the output variable.




## Parameters

Input

**IN1** (STRING)

Input string

**IN2** (STRING)

Replacement string

**L** (INT)

Number of characters in the input string to be replaced

**P** (INT)

Start position of the input string to be replaced, where 1 is the first character of the string

Output

**Unnamed output** (STRING)

Resulting string

## Remarks

- If this instruction is used with UTF-8 strings, please refer to the notes concerning UTF-8 strings under the data type STRING.
- If the strings are longer than the length defined for the input variables **input_string_1** and **input_string_2** in the field “Type”, an error occurs (see “Monitor” > “Special flags and registers” > “Basic error messages”

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if a string applied at the input or output is an invalid string

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if a string applied at the input or output is an invalid string

## Example

### POU header

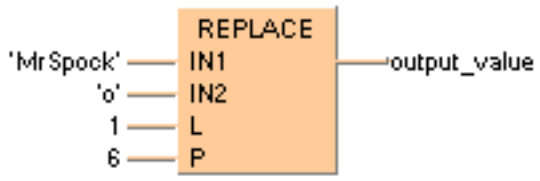
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | output_value | STRING[32] | '' | result: 'MrSpook' |

### POU body

In this example constant values are entered directly at the function's input contact pins. However, you may declare variables in the POU header. The STRING values have to be put in inverted commas, either in the POU header or at the contact pins. Here the 'c' in the STRING 'MrSpock' has been replaced with an 'o', yielding 'MrSpook'.

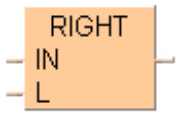
### LD body



# RIGHT

Copy characters from the right

RIGHT copies, starting from the right, **n** characters of the string of the first input variable to the output variable. You define the number of characters to be delivered **n** by the second input variable.




## Parameters

Input

**IN** (STRING)

1st input: input string

**L** (INT)

2nd input: number of input string's characters that are copied, from the right

Output

**Unnamed output** (STRING)

copied string

## Remarks

- If this instruction is used with UTF-8 strings, please refer to the notes concerning UTF-8 strings under the data type STRING.
- If the number of characters to be delivered is greater than the input string, the complete string will be copied to the output variable.
- If the output string is longer than the length defined for the output variable in the field “Type”, only as many characters are copied as the output variable can hold.The system variable **sys_bIsCarry** is set.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if a string applied at the input or output is an invalid string

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if a string applied at the input or output is an invalid string

**sys_blsCarry** (turns to TRUE for one scan)

if the output string is longer than the length defined for the output variable in the field "Type"

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

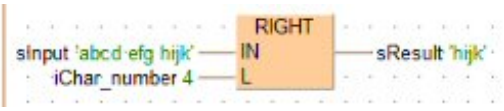
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 1 | VAR | sInput | STRING[15] | 'abcd efg hijk' | sample string |
| 2 | VAR | iChar_number | INT | 4 | characters to be delivered |
| 3 | VAR | sResult | STRING[4] | '' | result here= 'hijk' |

In this example the input variables (**sInput** and **iChar_number**) have been declared. Instead, you may enter the string ('abcd efg hijk') and the number of characters to be delivered directly into the function. The string has to be put in inverted commas, both in the POU header and in the function.

### POU body

Starting from the right, **iChar_number** (4) of **sInput** ('abcd efg hijk') is copied to **sResult** ('hijk').

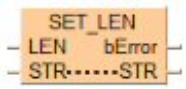
### LD body



# SET_LEN

Set string length

This instruction sets the string length of a string at the input **STR** to a new length defined at the input **LEN**. If the new length exceeds the maximum length of the string declaration, an error occurs and **bError** is set to TRUE. If you define a new string length that is shorter than the current string length, the string will be truncated to the new string length.




## Parameters

Input

**LEN** (INT)

New string length

Input/output

**STR** (STRING)

String set to the new length

Output

**bError** (BOOL)

Set to TRUE, if the new string length exceeds the maximum string length of string declaration

## Remarks

- If this instruction is used with UTF-8 strings, please refer to the notes concerning UTF-8 strings under the data type STRING.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

if a string applied at the input or output is an invalid string

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

if a string applied at the input or output is an invalid string

## Example

### POU header

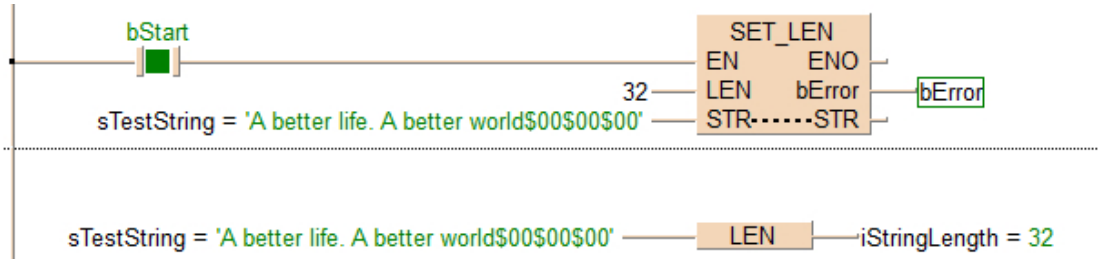
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | sTestString | STRING[128] | 'A better life. A better world' |
| 1 | VAR | iStringLength | INT | 0 |
| 2 | VAR | bError | BOOL | FALSE |
| 3 | VAR | bStart | BOOL | FALSE |

### POU body

When the variable **bStart** is set to TRUE, the function is carried out.

### LD body



## 32.13 FP instructions

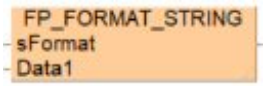
Tip

Advantages of FP instructions

# FP_FORMAT_STRING

Write formatted data into a result string

This instruction writes the formatted data specified at **Data1** (expandable to **Data16**) to a formatted string where the format is specified by the control string **sFormat**.




Input

**sFormat** (STRING)

Control string with format specifiers defining how the formatted string should be created

- String variable or character constant (up to 256 characters) containing the definition how the string should be created.
- The control string consists of the text, format specifier (%d,%e, etc.), line feed code (\n), and tab code (\t).
- The formatted string can contain up to 4096 characters. If it exceeds 4096 characters, an operation error will occur.
- You can specify up to 16 digits for a format specifier. If it exceeds 16 digits, an operation error will occur.
- The maximum number of characters after conversion of 1 data is 32, excluding %s and %S. If it exceeds 32 characters, an operation error will occur.
- All character strings that are not recognized as a format specifier are treated as text to be formatted.

**Data1** (INT, UINT, WORD, DINT, UDINT, DWORD, REAL, LREAL, STRING)

Data to be formatted

- Variable data to be written to the formatted string.
- Arrange the variable data in the order specified in the control string.

Output

**Result** (STRING)

Formatted string with data in ASCII format.

## Remarks

- The input **Data1** can be expanded to a maximum of 16 inputs.
- The number of inputs **Data1**..**DataN** must be the same as the number of format specifiers in the input string specified at **sFormat**.

- The data type at an input of **DataN** must correspond to the according format specifier in the **sFormat** (see table below).

### Explanation and examples of control string sFormat

**sFormat**: e.g. `%+12.5d,`

Use string data in the format shown below to specify the type, number of characters, and precision of the formatted string. A variety of options (such as inserting a sign or spaces) can also be selected depending on the type of data to be converted.

| Position in the example | Description | | sFormat | Binary data | Conversion result in ASCII data | Comment |
|---|---|---|---|---|---|---|
| '%+12.5d,' | **0** | zero padding | '%05d' | 100 | '00100' | when the number of characters is specified, zero padding can be used by adding zero (0) to the control string **sFormat** at this position. |
| | **+** | add a plus sign | '%+4d' | 100 | '+100' | width four characters, decimal value, + sign added |
| | **-** | left alignment | '%-6d' | 100 | '100␣␣␣' | width six characters, decimal value, left aligned |
| | ␣ | (space) add a space instead of plus sign | '%␣4d' | 100 | '␣100' | width four characters, decimal value filled with spaces |
| | # | insert 0x for hexadecimal numbers | '%#4X' | 100 | '0X64' | width four characters, numerical value in hexadecimal format starting with 0X |
| | | append always a decimal point for real number | '%#8.0f' | 123.45678 | '␣␣␣␣123.' | width eight characters, floating-point number, four leading spaces |

| Position in the example | Description | | sFormat | Binary data | Conversion result in ASCII data | Comment |
|---|---|---|---|---|---|---|
| '%+12.5d,' | **12** | width of the ASCII data element (with or without comma) When the width is smaller than the number of characters required by this value, an operation error occurs. Please refer to "Error flags". | '%012d' | 100 | '000000000100' | width 12 characters with nine leading zeros |
| | no width: | | '%d,' | 100 | '100,' | no width is specified, comma is appended |
| '%+12.5Ld,' | precision after decimal point | | | | | |
| | **.5** | any digit after decimal point | '%8.3f' | 123.45599 | '␣123.456' | width eight characters, three characters precision after decimal point |
| | **12.5** | specify the total number of characters (n) and the number of characters of precision (m) with [n.m], [n], or [.m]. The number of characters of precision (m) changes according to the type of conversion data. | | | | |
| | d , Ld, i , Li, u ,Lu, x , Lx , b, Lb | represents the number of characters in numerical strings. | | | | |
| | f, Lf, e, Le, E, LE | represents the number of characters after the decimal point. | | | | |
| | g, Lg, G, LG | represents the number of significant figures. | | | | |
| '%+12.5d,' | Type of data to be converted | | '%06d' | 100 | '␣␣␣100' | d , i , u , x , X , b: number of characters in numerical strings<br>f: number of characters after the decimal point.<br>g: number of significant figures. |

| Position in the example | Description | | sFormat | Binary data | Conversion result in ASCII data | Comment |
|---|---|---|---|---|---|---|
| | **d** | signed integer → decimal ASCII | '%6d' | 100 | '000100' | width six characters, decimal value with three leading zeros |
| | **u** | unsigned integer → decimal ASCII | '%10u' | -100 | '␣␣␣␣␣␣␣100' | width 10 characters, seven leading spaces, unsigned integer |
| | **x** | unsigned integer → hexadecimal ASCII | '%4x' | 16#12A | '␣12a' | width four characters, hexadecimal number in lower case |
| | **X** | hexadecimal upper case | '%4X' | 16#12A | '␣12A' | width four characters, hexadecimal number in upper case |
| | **b** | BCD integer → hexadecimal ASCII | '%5b' | 16#123 | '␣123' | width five characters, BCD data |
| | **f** | floating point real number → floating point ASCII | '%-6.2f' | 1.2345 | '1.23␣' | width six characters, left aligned, precision two digits after decimal point |
| | **e** | floating point real number → exponential notation ASCII | '%9.3e' | 1234.5678 | '1.235e+03' | width 9 characters, precision 3 digits after decimal point, exponential lower case |
| | **E** | exponential upper case 1.23E10 | '%9.3E' | 1234.5678 | '1.235E+03' | width 9 characters, precision 3 digits after decimal point, exponential upper case |
| | **g** | floating point real number → floating point ASCII or exponential notation ASCII | '%12g' | 1234.5678 | '␣␣␣␣␣1234.57' | width 12 characters, floating-point number |

| Position in the example | Description | | sFormat | Binary data | Conversion result in ASCII data | Comment |
|---|---|---|---|---|---|---|
| | **G** | floating or exponential upper case | '%#9.3G' | 1234 | '␣␣␣1.E+03' | width nine characters, precision three characters after decimal point |
| | **s** | string →ASCII | '%10s' | abcdef | 'abcdef␣␣␣␣' | string data (for the specified number of characters) |
| | **S** | string →ASCII upper case | '%-10S' | abcDEF | '␣␣␣␣abcDEF' | string data case-sensitive |

## Conversion data for the control string sFormat

| Control string | Binary data before conversion | ASCII data after conversion | Example |
|---|---|---|---|
| '%d' or '%i' | 16-bit data (signed integer) | Decimal ASCII data | '%d' , '%5d' , '%+5d" '%-5d' , '%05d' , '%10.5d' , '%d,' , '% d' |
| '%Ld' or '%Li' | 32-bit data (signed integer) | | |
| '%u' | 16-bit data (unsigned integer) | | '%u' , '%5u' , '%+5u' '%-5u' , '%05u' , '%10.5u' , '%u,' |
| '%Lu' | 32-bit data (unsigned integer) | | |
| '%x' | 16-bit data | Hexadecimal ASCII data (forward/reverse direction) | '%x' , '%5x' , '%-5x' '%05x' , '%10.5x' , '%x,' , '%#x' , '%X' |
| '%Lx' | 32-bit data | | |
| '%b' | 16-bit BCD data | | '%b' , '%5b' , '%-5b' '%05b' , '%10.5b' , '%b,' |
| '%Lb' | 32-bit BCD data | | |
| '%f' | 32-bit single-precision real number data | Floating point ASCII data | '%f' , '%5.2f' , '%+5.2f' '%-5.2f' , '%05.2f' , '%f,' , '%#f' , '% f' |
| '%Lf' | 64-bit double-precision real number data | | |
| '%e' | 32-bit single-precision real number data | Exponential notation ASCII data | '%e' , '%5.2e' , '%+5.2e' '%-5.2e' , '%05.2e' ,'%e,' '%#5.2e' , '% e' , '%E' |
| '%Le' | 64-bit double-precision real number data | | |
| '%LE' | 64-bit double-precision long real number data | | '%#5.2E' , '% E' , '%E' |
| '%g' | 32-bit single-precision real number data | Exponential notation ASCII data or floating point ASCII data (whichever is shorter in the relevant notation) | '%g' , '%5.2g' , '%+5.2g' '%-5.2g' , '%05.2g' ,'%g," '%#5.2g' , '%G' |
| '%Lg' | 64-bit double-precision real number data | | |

| Control string | Binary data before conversion | ASCII data after conversion | Example |
|---|---|---|---|
| '%LG' | 64-bit double-precision long real number data | | '%-5.2G' , '%05.2G' ,'%G," '%#5.2G' , '%G' |
| '%s' | String data | String data (for the specified number of characters) | '%s' , '%5s' , '%-5s' , '%-05s' |

## Error flags

**sys_blsOperationErrorHold** (turns to TRUE and remains TRUE)

- if there is an error in the control string specified by **sFormat**.
- if forward direction (+) is specified in **sFormat** when the format is decimal.
- if the converted result exceeds the area.
- if a width of the ASCII data element is specified which is smaller than the number of characters required by the value

**sys_blsOperationErrorNonHold** (turns to TRUE for one scan)

- if there is an error in the control string specified by **sFormat**.
- if forward direction (+) is specified in **sFormat** when the format is decimal.
- if the converted result exceeds the area.
- if a width of the ASCII data element is specified which is smaller than the number of characters required by the value

## Example

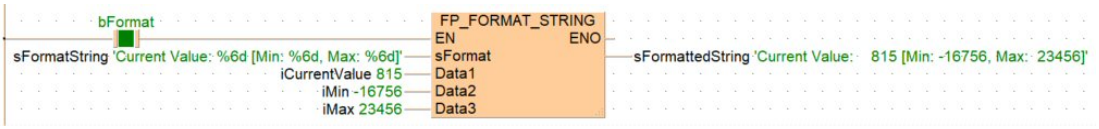
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 1 | VAR | bFormat | BOOL | FALSE |
| 2 | VAR | sFormatString | STRING[50] | 'Current Value: %6d [Min: %6d, Max: %6d]' |
| 3 | VAR | sFormattedString | STRING[50] | '' |
| 4 | VAR | iCurrentValue | INT | 815 |
| 5 | VAR | iMin | INT | -16756 |
| 6 | VAR | iMax | INT | 23456 |

### LD body

When the variable **bStart** changes from FALSE to TRUE, the function is carried out.


bFormat
FP_FORMAT_STRING
EN
ENO
sFormatString 'Current Value: %6d [Min: %6d, Max: %6d]'
sFormat
sFormattedString 'Current Value: 815 [Min: -16756, Max: 23456]'
iCurrentValue 815
Data1
iMin -16756
Data2
iMax 23456
Data3

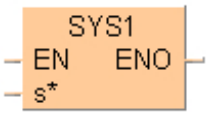
# 33 System register instructions

# SYS1

Change PLC system setting

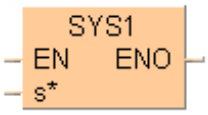
The description for **SYS1** is divided into the following sections:


- **SYS1** Communication condition setting for the COM ports of the CPU (page 1861)
- **SYS1** Password setting (page 1866)
- **SYS1** Interrupt setting (page 1868)
- **SYS1** PLC link time setting (page 1870)
- **SYS1** Change high-speed counter operation mode (page 1873)
- **SYS1** RS485 response time control (page 1875)

# SYS1 Communication condition setting for the COM ports of the CPU

This changes the communication conditions for the COM port or Tool port based on the contents specified by the character constant.




## Remarks

- Executing this instruction does not rewrite the contents of the system ROM in the control unit. As a result, turning the power supply off and then on again rewrites the contents of the system registers specified by the tool software.
- We recommend using differential execution with this instruction.
- Because the system register settings are changed, a verification error may occur in some cases if verification is carried out with the tools.
- Separate first and second keywords with a comma "," and do not use spaces.
- The communication conditions for the port specified by the first keyword are changed to the contents specified by the second keyword. The first and second keywords are separated by a comma.

  Contents that can be changed include the following:

  1. Communication format
  2. Baud rate
  3. Unit No.
  4. Header and Terminator
  5. RS (Request to Send) control

## Keyword setting

1. Communication format (Shared by the Tool, COM 1 and COM 2 ports)

   ```
   TOOL, B7PNS1
   ```

| | |
|---|---|
| TOOL | Port used<br>TOOL: Tool port<br>COM1: COM1 port<br>COM2: COM2 port |
| Character bit | B7: 7 bits<br>B8: 8 bits |

| | |
|---|---|
| Parity | PN: none<br>PO: odd parity<br>PE: even parity |
| Stop bit | S1: stop bit 1<br>S2: stop bit 2 |

2. Baud rate (Shared by the Tool, COM 1 and COM 2 ports)

```
TOOL, 19200
```

| | |
|---|---|
| TOOL | Port used<br>TOOL: Tool port<br>COM1: COM1 port<br>COM2: COM2 port |
| Baud rate | 2400: 2,400 bps<br>4800: 4,800 bps<br>9600: 9,600 bps<br>19200: 19,200 bps<br>38400: 38,400 bps<br>57600: 57,600 bps<br>115200: 115,200 bps |

Lower baud rates of 300, 600, and 1200bit/s can be specified for FP-X V2.0 or later and FPΣV3.1 or later. These baud rates cannot be set in the system registers.

3. Unit No. (Shared by the Tool, COM 1 and COM 2 ports)

```
COM1, No1
```

| | |
|---|---|
| COM1 | Port used<br>TOOL: Tool port<br>COM1: COM1 port<br>COM2: COM2 port |
| No1 | Unit number<br>No1–No99 (n: 1–99) |

With the FP0R, use the keywords 'COM1No' and 'TOOLNo' to read the unit number from a data register (DT0–DT9999) containing the unit number 1–99. The data register has to be specified with exactly five characters: For example, D0815 indicates DT815. Leading zeros must be entered. The keyword is case sensitive, hence COM1NO, Com1No or … d0815 would be invalid.

Example

- SYS1 'COM1No,D9999' indicates DT9999
- SYS1 'COM1No,D0000' indicates DT0
- A calculation error occurs if any value except 1–99 is assigned to the DT memory.

4. Header and Terminator (Shared by the COM 1 and COM 2 ports)

```
COM1, STX
```

| | |
|---|---|
| COM1 | Port used<br>COM1: COM1 port<br>COM2: COM2 port |
| Header | STX: use STX<br>NOSTX: no STX (n: 1–99) |
| Terminator | ETX: use ETX<br>CR: use CR<br>CRLF: use CR and LF<br>NOTERM: none |

5. RS (Request to Send) control (COM 1 port only)

```
COM1, RTS1
```

| | |
|---|---|
| COM1 | Port used<br>COM1: COM1 port |
| RTS1 | RS control for the 1-channel RS232C type communication cassette<br>RTS1: Disables communication (Sets the RS terminal to “on”)<br>RTS": Enables communication (Sets the RS terminal to “off”) |

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if any character other than a keyword is specified
- if no comma is between the first and second keywords
- if small letters of the alphabet are used to specify the keyword (except for numbers used to specify unit no.)
- if no communication cassette has been installed when COM1 or COM2 has been set
- if the setting of the unit no. setting switch is anything other than 0 when COM1 or COM2 has been set and the unit no. is being changed
- if the unit no. set using this instruction is anything other than a value between 1 and 99
- if the baud rate or transmission format for COM1 has been changed when the PLC link mode is specified for COM1
- if the baud rate or transmission format is changed while the Tool port, COM port 1, or COM port 2 is being initialized using MODEM
- if the communication mode is set to anything other than the general communication mode when header and terminator have been set
- if any communication cassette other than the 1-channel RS232C type communication cassette is installed when using RS control
- if the specified unit no. is larger than the largest unit no. specified by the system register when the COM 1 port is in the PLC link mode

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if any character other than a keyword is specified
- if no comma is between the first and second keywords
- if small letters of the alphabet are used to specify the keyword (except for numbers used to specify unit no.)
- if no communication cassette has been installed when COM1 or COM2 has been set
- if the setting of the unit no. setting switch is anything other than 0 when COM1 or COM2 has been set and the unit no. is being changed
- if the unit no. set using this instruction is anything other than a value between 1 and 99
- if the baud rate or transmission format for COM1 has been changed when the PLC link mode is specified for COM1
- if the baud rate or transmission format is changed while the Tool port, COM port 1, or COM port 2 is being initialized using MODEM
- if the communication mode is set to anything other than the general communication mode when header and terminator have been set
- if any communication cassette other than the 1-channel RS232C type communication cassette is installed when using RS control
- if the specified unit no. is larger than the largest unit no. specified by the system register when the COM 1 port is in the PLC link mode

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.

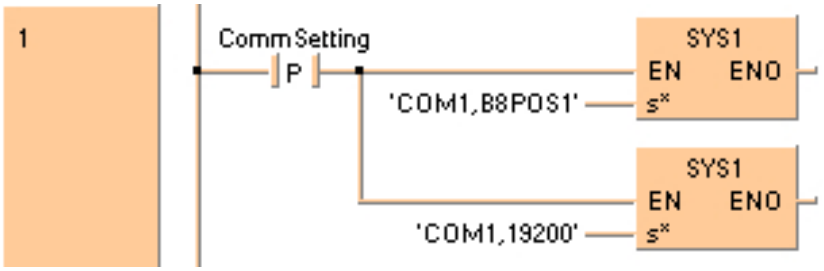
The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | CommSettings | BOOL | FALSE | Sets transmission format to: Character bit 8. Parity: Odd. Stop bit: 1. Baud rate: 19,200 bps |

### POU body

When **CommSettings** turns on, the transmission format and baud rate for the COM1 port are set as follows: Character bit: 8, Parity: Odd; Stop bit: 1; Baud rate: 19,200 bps.

### LD body



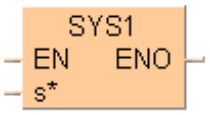

Note

The values entered at **s*** will be right aligned automatically by the compiler.

# SYS1 Password setting

This changes the password specified by the controller, based on the contents specified by the character constant.

This changes the password specified by the controller to the contents specified by the second keyword. The first and second keywords are separated by a comma.




## Remarks

- When this instruction is executed, writing to the internal FROM takes approximately 100ms.
- If the specified password is the same as the password that has already been written, the password is not written to the FROM.
- We recommend using differential execution with this instruction.
- Separate first and second keywords with a comma "," and do not use spaces.
- Keyword setting for 4-digit hexadecimal password

  ```
  PASS,ABCD
  ```

  | | |
  |---|---|
  | PASS | fixed |
  | ABCD | Password, e.g. set password to ABCD |

- Keyword setting for 8-digit alphanumeric password

  Enter for example 'PAS,FP-X␣v␣3'. Spaces at the end of the password are not significant.

  ```
  PAS,FP-X v 3
  ```

  | | |
  |---|---|
  | PAS | fixed |
  | FP-X v 3 | Password, e.g. set password to FP-X v 3 |

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if any character other than a keyword is specified
- if no comma is between the first and second keywords
- if small letters of the alphabet are used to specify the keyword
- if the data specified for the password setting is any character other than 0 to 9 or A to F, or the specified data consists of other than four digits.

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if any character other than a keyword is specified
- if no comma is between the first and second keywords
- if small letters of the alphabet are used to specify the keyword
- if the data specified for the password setting is any character other than 0 to 9 or A to F, or the specified data consists of other than four digits.

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header.

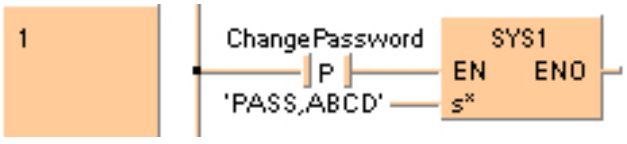
The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | ChangePassword | BOOL | FALSE |

### POU body

When **ChangePassword** turns on, the controller password is changed to "ABCD".

### LD body



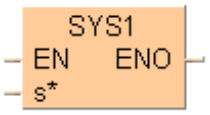

Note

The values entered at **s*** will be right aligned automatically by the compiler.

# SYS1 Interrupt setting

This sets the interrupt input based on the contents specified by the character constant.

This sets the input specified by the first keyword as the interrupt input, and changes the input conditions to the contents specified by the second keyword. The first and second keywords are separated by a comma.




## Keyword setting

```
INT2,UP
```

| | |
|---|---|
| INT2 | Interrupt Input<br>INT0–INT7: X0–X7 |
| UP | Effective edges<br>UP: Rising edge<br>DOWN: Falling edge<br>BOTH: Rising and falling edged |

For the FP-X you can set INT0–INT13.

## Remarks

- Executing this instruction does not rewrite the contents of the system ROM in the control unit. As a result, turning the power supply off and then on again rewrites the contents of the system registers specified by the tool software.
- We recommend using differential execution with this instruction.
- When UP or DOWN has been specified, the contents of the system registers change in accordance with the specification, so a verification error may occur in some cases, when the program is verified. When BOTH has been specified, the contents of the system registers do not change.
- Separate first and second keywords with a comma "," and do not use spaces.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if any character other than a keyword is specified
- if no comma is between the first and second keywords
- if small letters of the alphabet are used to specify the keyword

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if any character other than a keyword is specified
- if no comma is between the first and second keywords
- if small letters of the alphabet are used to specify the keyword

## Example

### POU header

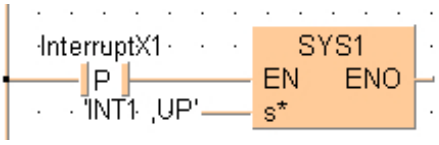
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | InterruptX1 | BOOL | FALSE |

### POU body

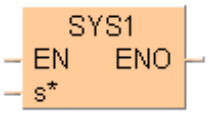
When **InterruptX1** turns on, the input condition of interrupt input X1 is changed to "Rising Edge".

### LD body



# SYS1 PLC link time setting

This sets the system setting time when a PLC link is used, based on the contents specified by the character constant.




## Remarks

- The program should be placed at the beginning of all PLCs being linked, and the same values specified.
- This instruction should be specified in order to set special internal flag R9014 as the differential execution condition.
- The setting contents of the system registers are not affected by this instruction being executed.
- Separate first and second keywords with a comma "," and do not use spaces.
- Precautions when setting the link entry wait time
  - This should be specified such that the value is at least twice that of the largest scan time of all the PLCs that are linked.
  - If a short value has been specified, there may be some PLCs that are not able to join the link even though the power supply for that PLC has been turned on.
  - If there are any stations that have not joined the link, the setting should not be changed, even if the link transmission cycle time is longer as a result. (The default value is 400 ms.)
- Precautions when setting the error detection time for the transmission assurance flag
  - This should be specified such that the value is at least twice that of the largest transmission cycle time of all the PLCs that are linked.
  - If a short value has been specified, there is a possibility that the transmission assurance flag will malfunction.
  - The setting should not be changed, even if the detection time for the transmission assurance flag is longer than the result. (The default value is 6400ms.)
- The conditions specified by the first keyword are set as the time specified by the second keyword. The first and second keywords are separated by a comma.
- The setting for the link entry waiting time is set if the transmission cycle time is shortened when there are stations that have not joined the link. (Stations that have not joined the link: Stations that have not been connected between the first station and the station with the largest number, or stations for which the power supply has not been turned on.)
- The error detection time setting for the transmission assurance flag is set if the time between the power supply being turned off at one station and the transmission assurance flag being turned off at a different station is to be shortened.

### Keyword setting

1. Link entry wait time

   `PCLK1T0,100`

   | | |
   |---|---|
   | PCLK1T0 | fixed |
   | 100 | Specified range:<br>10–400 (10–400 ms) |

2. Error detection time for transmission assurance flag

   `PCLK1T1,100`

   | | |
   |---|---|
   | PCLK1T1 | fixed |
   | 100 | Specified range:<br>100–6400 (100–6400 ms) |

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if any character other than a keyword is specified
- if no comma is between the first and second keywords
- if small letters of the alphabet are used to specify the keyword
- if the specified value is outside the specified range

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if any character other than a keyword is specified
- if no comma is between the first and second keywords
- if small letters of the alphabet are used to specify the keyword
- if the specified value is outside the specified range

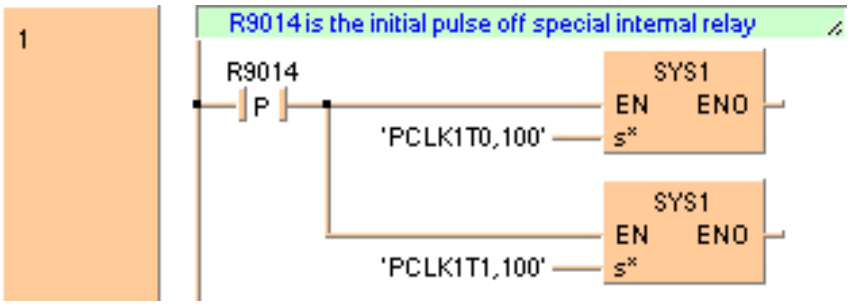
## Example

Because FP addresses and strings are entered directly instead of using variables, no POU header is required.

When **sys_bIsNotFirstScan** turns on when a PLC link is being used, the link entry wait time and the error detection times for transmission assurance flag are set as follows:

- Link entry wait time: 100ms
- Error detection time for transmission assurance flag: 100ms.

### LD body

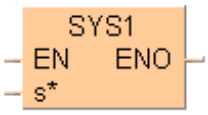



Note

The values entered at **s*** will be right aligned automatically by the compiler.

# SYS1 Change high-speed counter operation mode

This changes the operation mode of the high-speed counter based on the contents specified by the character constant.




## Keyword setting

`HSCn,UP`

| | |
|---|---|
| HSCn | High-speed counter setting<br>n: 0–9, A, B (FP-X C14R, C30/60R)<br>n: 0–7 (FP-X C14T, C30/60T)<br>n: 0–3 (FPΣ) |
| UP | UP: addition input setting<br>DOWN: subtraction input setting |

Example: `HSC1,UP`

## Remarks

- If the corresponding HSC system register is set to Unused, an operation error occurs. Set the system register to Incremental input or Decremental input in advance.
- Executing this instruction does not rewrite the contents of the system ROM in the control unit. As a result, turning the power supply off and then on again rewrites the contents of the system registers specified by the software tool.
- We recommend to execute this instruction only once, e.g. in dependency of a rising or falling edge of an execution condition.
- When UP or DOWN has been specified, the contents of the system registers change in accordance with the specification, so a verification error may occur in some cases when checking or compiling the program. When BOTH have been specified, the contents of the system registers do not change. Separate the first and the second keyword with a comma "," e.g. **HSCB,UP**; do not use spaces. Otherwise an operation error will occur.

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- If something that is not identical with a keyword is specified
- If there is no comma between the first and the second keyword
- If the letters used to specify the keyword are not capitalized

- If the HSC system register is set to items other than the addition input or subtraction input

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if any character other than a keyword is specified
- If there is no comma between the first and the second keyword
- If the letters used to specify the keyword are not capitalized
- If the HSC system register is set to items other than the addition input or subtraction input

## Example

### POU header

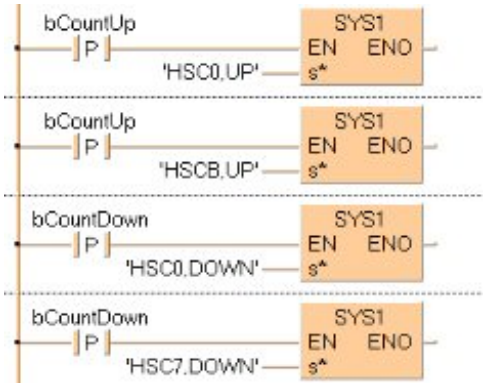
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | bCountUp | BOOL | FALSE |
| 1 | VAR | bCountDown | BOOL | FALSE |

### POU body

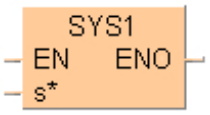
When **bCountUp** is set to TRUE, the function is carried out. The system register for the specified channel is set to count up. When **bCountDown** is set to TRUE, the specified channel is set to count down.

### LD body



# SYS1 RS485 response time control

This changes the communication conditions based on the RS485 of the COM port or Tool port, in response to the contents specified by the character constant.

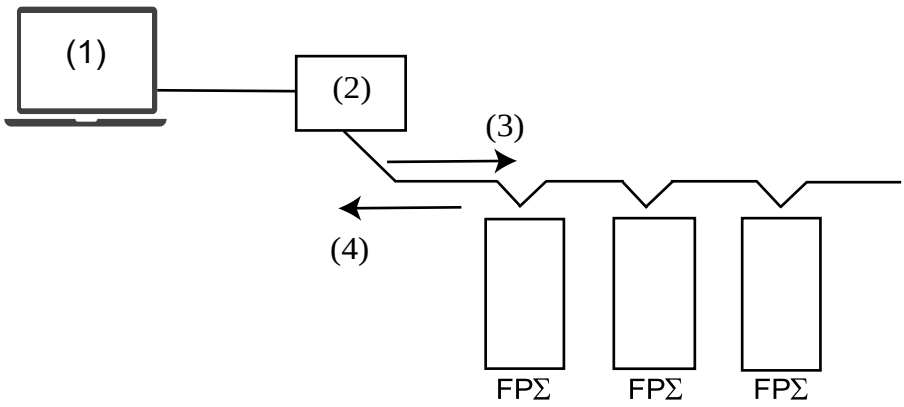



## Remarks

- This instruction is valid only if the setting on the controller side has been set to the computer link mode or the PLC link mode. It is invalid in the general communication mode.
- Executing this instruction does not change the settings in the system registers.
- We recommend using differential execution with this instruction.
- When the power supply to the PLC is off, the settings set by this instruction are cleared. (The set value will become 0.) If the mode is switched to the PROG mode after the instruction has been executed, however, the settings will be retained.
- If a commercial RS232C/RS485 converter is being used in the PLC link mode, this instruction should be programmed in all of the stations (PLCs) connected to the link.
- Separate first and second keywords with a comma "," and do not use spaces.
- The port response time specified by the first keyword is delayed based on the contents specified by the second keyword. This instruction is used to delay the response time on the PLC side until the state is reached in which commands can be sent by an external device and responses can be received from the PLC.
- The first and second keywords are separated by a comma.

### Example

When a commercial RS232C/RS485 converter is being used to carry out communication between a personal computer and the FP-∑, this instruction is used to return the PLC response after switching of the enable signal has been completed on the converter side.




(1) External device (PC)
(2) Commercial RS232C/RS485 converter
(3) Command
(4) Response

## Keyword setting

```
TOOL,WAITn
```

| | |
|---|---|
| TOOL | Port used<br>TOOL: Tool port<br>COM1: COM1 port<br>COM2: COM2 port |
| WAITn | Response time<br>WAIT0–WAIT999 (n: 0–999) |

- If the communication mode has been set to the computer link mode, the set time is the scan time x **n** (n: 0 to 999).
- If the communication mode has been set to the PLC link mode, the set time is **n**μs (n: 0 to 999).
- If **n** = 0, the delay time set by this instruction will be set to "None".

## Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if any character other than a keyword is specified
- if no comma is between the first and second keywords
- if small letters of the alphabet are used to specify the keyword
- if no communication cassette has been installed when COM1 or COM2 has been set

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if any character other than a keyword is specified
- if no comma is between the first and second keywords
- if small letters of the alphabet are used to specify the keyword

- if no communication cassette has been installed when COM1 or COM2 has been set

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | ChangeResponseT | BOOL | FALSE | Changes response time of RS485 |

### POU body

When **ChangeResponseT** turns on, the response time for COM port 1 is delayed by 2μs.

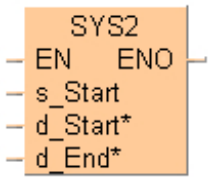
### LD body


Note

The values entered at **s*** will be right aligned automatically by the compiler.

# SYS2

Change system register settings for PC link area




You can change the values in system registers 40 - 47 (with the FP0R, FP-Σ 32k, FP-X also 50 - 57), PC link area.

While the PLC is in RUN mode, **SYS2** changes the settings for the specified system registers. **s_Start** contains the new values for those system registers defined between **d_Start** and **d_End**.

## Remarks

- Executing this instruction does not rewrite the contents of the system ROM in the control unit. As a result, turning the power supply off and then on again rewrites the contents of the system registers specified by the tool software.
- A value between 40 and 47 should be specified for **d_Start** or **d_End**. Also, the values should always be specified in such a way that **d_Start**≤**d_End**.
- The values of the system registers change, so a verification error may occur when the program is verified.

## Parameters

Input

**s_Start** (WORD, INT, UINT)

Contains new values for the system registers defined by remaining two variables.

**d_Start** (WORD, INT, UINT)

First system register (between 40-47) to receive new value.
must be a constant

**d_End** (WORD, INT, UINT)

Last system register (between 40-47) to receive new value.
must be a constant

### Error flags

**sys_bIsOperationErrorHold** (turns to TRUE and remains TRUE)

- if **d_Start** > **d_End**
- if the specified value is outside the ranges specified for the various system registers setting values

**sys_bIsOperationErrorNonHold** (turns to TRUE for one scan)

- if **d_Start** > **d_End**
- if the specified value is outside the ranges specified for the various system registers setting values

## Example

### DUT

With a Data Unit Type (DUT) you can define a data unit type that is composed of other data types. A DUT is first defined in the DUT pool and then processed like the standard data types (BOOL, INT, etc.) in the list of global variables or the POU header.

LINK_AREAS [DUT]

| | Identifier | Type | Initial | Comment |
|---|---|---|---|---|
| 0 | RelayArea | INT | 0 | System register 40 |
| 1 | RegisterArea | INT | 0 | System register 41 |
| 2 | RelaySendStart | INT | 0 | System register 42 |
| 3 | RealySendSize | INT | 0 | System register 43 |
| 4 | RegisterSendStart | INT | 0 | System register 44 |
| 5 | RegisterSendSize | INT | 0 | System register 45 |

### POU header

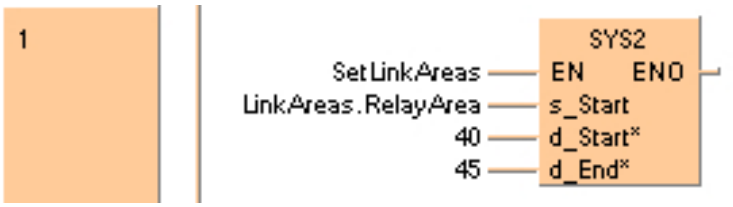
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | LinkAreas | LINK_AREAS | RelayArea := 64, |
| 1 | VAR | SetLinkAreas | BOOL | FALSE |

### POU body

Changes the values for the PC link area system registers 40 through 45 as defined in **LinkAreas** when **SetLinkAreas** turns on.

### LD body

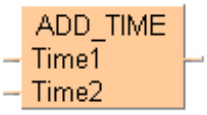

1
SYS2
EN
ENO
SetLinkAreas
LinkAreas.RelayArea
s_Start
40
d_Start*
45
d_End*

# 34 Timer instructions

# ADD_TIME

Add TIME

**ADD_TIME** adds the times of the two input variables and writes the sum in the output variable.




## Parameters

Input

**Time1** (DATE_AND_TIME)

1st input: addend

**Time2** (TIME)

2nd input: addend

Output

**VAR_OUT** (DATE_AND_TIME)

sum

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | time_value_1 | TIME | T#0s |
| 1 | VAR | time_value_2 | TIME | T#0s |
| 2 | VAR | time_value_3 | TIME | T#0s |

In this example the input variables **time_value_1** and **time_value_2** have been declared. Instead, you may enter constants directly at the input contacts of a function.

### POU body

**time_value_1** and **time_value_2** are added. The result is written into **time_value_3**.

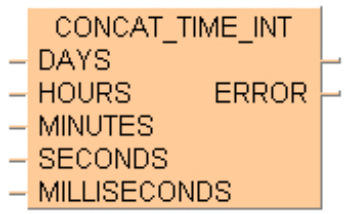
**LD body**


ADD_TIME
time_value_1 = T#4s400ms
time_value_2 = T#5s600ms
Time1
Time2
time_value_3 = T#10s

# CONCAT_TIME_INT

Concatenate INT values to form a time

**CONCAT_TIME_INT** concatenates the INTEGER values for days, hours, minutes, seconds, and milliseconds.The result is stored in the output variable of the data type TIME.The Boolean output ERROR is set if the input values are invalid date or time values.




## Parameters

Input

**DAYS** (INT)

1st input: days

**HOURS** (INT)

2nd input: hours

**MINUTES** (INT)

3rd input: minutes

**SECONDS** (INT)

4th input: seconds

**MILLISECONDS** (INT)

5th input: milliseconds

Output

**VAR_OUT** (TIME)

Result

**ERROR** (BOOL)

The Boolean output ERROR is set if the input values are invalid date or time values.

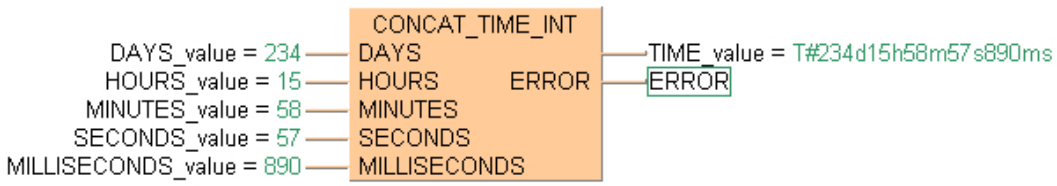
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | TIME_value | TIME | T#0s |
| 1 | VAR | DAYS_value | INT | 234 |
| 2 | VAR | HOURS_value | INT | 15 |
| 3 | VAR | MINUTES_value | INT | 58 |
| 4 | VAR | SECONDS_value | INT | 57 |
| 5 | VAR | MILLISECONDS_value | INT | 890 |
| 6 | VAR | ERROR | BOOL | FALSE |

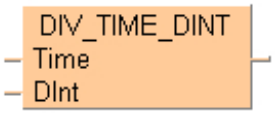
### LD body



# DIV_TIME_DINT

Divide TIME by DOUBLE INTEGER

**DIV_TIME_DINT** divides the value of the first input variable by the value of the second and writes the result into the output variable.




## Parameters

Input

**TIME** (TIME)

1st input: dividend

**DINT** (DINT)

2nd input: divisor

Output

**TIME** (TIME)

result

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

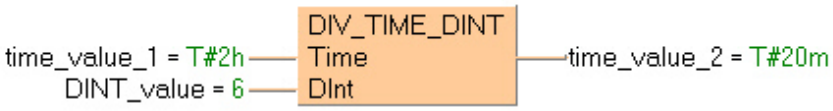
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | time_value_1 | TIME | T#2h | |
| 1 | VAR | time_value_2 | TIME | T#0s | result: T#20m |
| 2 | VAR | DINT_value | DINT | 6 | |

In this example, the input variables **time_value_1**, **DINT_value** have been declared. However, you can write a constant directly at the input contact of the function instead.

### POU body

**time_value_1** is divided by **DINT_value**. The result is written in **time_value_2**.

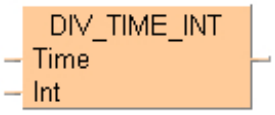
### LD body


DIV_TIME_DINT
time_value_1 = T#2h
Time
DINT_value = 6
DInt
time_value_2 = T#20m

# DIV_TIME_INT

Divide TIME by INTEGER

**DIV_TIME_INT** divides the value of the first input variable by the value of the second input variable and writes the result into the output variable.




## Parameters

Input

**Time** (TIME)

1st input: dividend

**Int** (INT)

2nd input: divisor

Output

**Time** (TIME)

result

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

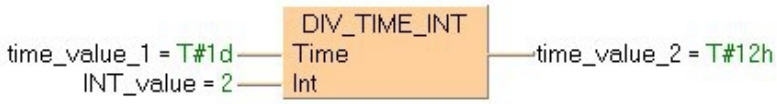
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | time_value_1 | TIME | T#0s |
| 1 | VAR | time_value_2 | TIME | T#0s |
| 2 | VAR | INT_value | INT | 0 |

In this example the input variables **time_value_1** and **INT_value** have been declared. Instead, you may enter constants directly at the input contacts of a function.

### POU body

**time_value_1** is divided by **INT_value**. The result is written into **time_value_2**.

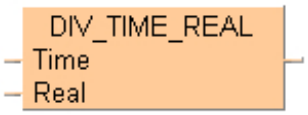
### LD body


DIV_TIME_INT
time_value_1 = T#1d
Time
INT_value = 2
Int
time_value_2 = T#12h

# DIV_TIME_REAL

Divide TIME by REAL

**DIV_TIME_REAL** divides the value of the first input variable of the data type TIME by the value of the second input variable of the data type REAL. The REAL value is rounded off to the nearest whole number. The result is written into the output variable.




## Parameters

Input

**Time** (TIME)

1st input: dividend

**Real** (REAL)

2nd input: divisor

Output

**Time** (TIME)

result

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | input_time | TIME | T#10s |
| 1 | VAR | input_real | REAL | 2.4 |
| 2 | VAR | div_result | TIME | T#0s |

### POU body

The value of variable **input_time** is divided by the value of the variable **input_real**. The result is written in **div_result**. In this example the input variables have been declared in the POU header. However, you may enter constants directly at the contact pins of the function.

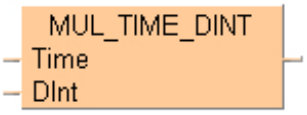
### LD body


DIV_TIME_REAL
input_time = T#10s
input_real = 2.4000001
Time
Real
div_result = T#4s170ms

# MUL_TIME_DINT

Multiply TIME by DOUBLE INTEGER

**MUL_TIME_DINT** multiplies the values of the input variables and writes the result to the output variable.




## Parameters

Input

**Time** (TIME)

1st input: multiplicand

**Dint** (DINT)

2nd input: multiplicator

Output

**Time** (TIME)

result

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

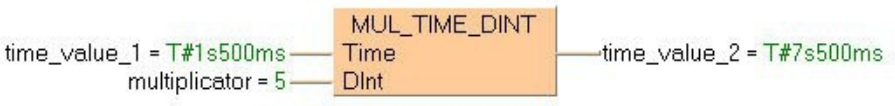
| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | time_value_1 | TIME | T#1s500ms | |
| 1 | VAR | multiplier | DINT | 5 | |
| 2 | VAR | time_value_2 | TIME | T#0s | result: T#7s500ms |

In this example, the input variables **time_value** and **multiplier** have been declared. However, you can write a constant directly at the input contact of the function instead.

### POU body

**time_value_1** is multiplied by **multiplier**. The result is written in **time_value_2**.

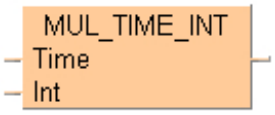
### LD body


MUL_TIME_DINT
time_value_1 = T#1s500ms
Time
multiplicator = 5
DInt
time_value_2 = T#7s500ms

# MUL_TIME_INT

Multiply TIME by INTEGER

**MUL_TIME_INT** multiplies the values of the two input variables with each other and writes the result into the output variable.




## Parameters

Input

**Time** (TIME)

1st input: multiplicand

**Int** (INT)

2nd input: multiplicator

Output

**Time** (TIME)

result

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

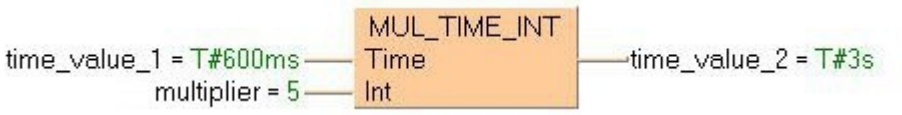
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | time_value_1 | TIME | T#0s |
| 1 | VAR | multiplier | INT | 0 |
| 2 | VAR | time_value_2 | TIME | T#0s |

In this example the input variables **time_value_1** and **multiplier** have been declared. Instead, you may enter constants directly at the input contacts of a function.

### POU body

**time_value_1** is multiplied with **multiplier**. The result is written into **time_value_2**.

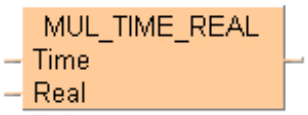
**LD body**


MUL_TIME_INT
time_value_1 = T#600ms
Time
multiplier = 5
Int
time_value_2 = T#3s

# MUL_TIME_REAL

Multiply TIME by REAL

**MUL_TIME_REAL** multiplies the value of the first input variable of the data type **TIME** by the value of the second input variable of the data type **REAL**. The **REAL** value is rounded off to the nearest whole number. The result is written into the output variable.




## Parameters

Input

**Time** (TIME)

1st input: multiplicand

**Real** (REAL)

2nd input: multiplicator

Output

**Time** (TIME)

result

## Example

### POU header

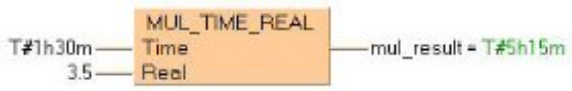
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | mul_result | TIME | T#0s |

### POU body

The constant T#1h30m is multiplied by the value 3.5. The result is written in **mul_result**. By clicking on the (Monitoring) icon while in the online mode, you can see the resultT#5h15m0s0.00ms immediately.

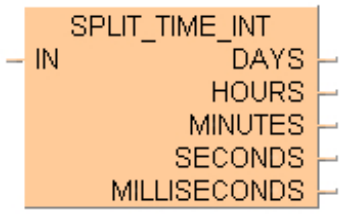
### LD body



# SPLIT_TIME_INT

Split TIME into INT values

**SPLIT_TIME_INT** splits a value of the data type TIME into INTvalues for days, hours, minutes, seconds, and milliseconds.




## Parameters

Input

**IN** (TIME)

time

Output

**DAYS** (INT)

1st output: days

**HOURS** (INT)

2nd output: hours

**MINUTES** (INT)

3rd output: minutes

**SECONDS** (INT)

4th output: seconds

**MILLISECONDS** (INT)

5th output: milliseconds

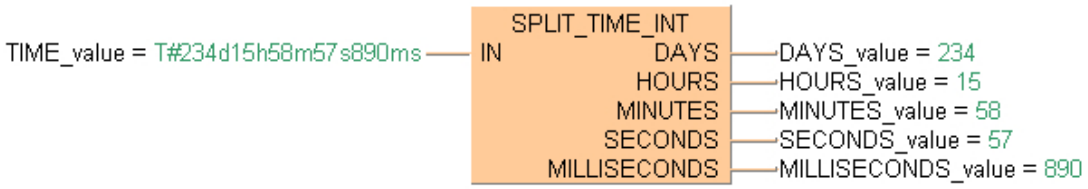
## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | TIME_value | TIME | T#234d15h58m57s890ms |
| 1 | VAR | DAYS_value | INT | 0 |
| 2 | VAR | HOURS_value | INT | 0 |
| 3 | VAR | MINUTES_value | INT | 0 |
| 4 | VAR | SECONDS_value | INT | 0 |

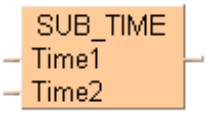
### LD body



# SUB_TIME

Subtract TIME

**SUB_TIME** subtracts the value of the second input variable from the value of the first input variable and writes the result into the output variable.




## Parameters

Input

**Time1** (TIME)

1st input: minuend

**Time2** (TIME)

2nd input: subtrahend

Output

**Time1** (TIME)

Result

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

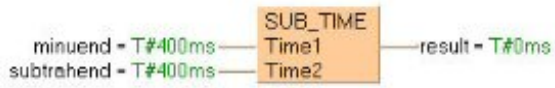
| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | minuend | TIME | T#0s |
| 1 | VAR | subtrahend | TIME | T#0s |
| 2 | VAR | result | TIME | T#0s |

In this example the input variables (**minuend** and **subtrahend**) have been declared. Instead, you may enter constants directly at the input contacts of a function.

### POU body

**Subtrahend** is subtracted from **minuend**. The **result** will be written into **result**.

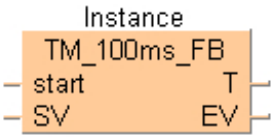
### LD body


SUB_TIME
minuend = T#400ms
Time1
subtrahend = T#400ms
Time2
result = T#0ms

# TM_100ms_FB

Timer for 100ms intervals (0 to 3276.7s)

This timer for 0.1s units works as an ON-delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.




## Parameters

Input

**start** (BOOL)

start contact
each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

**SV**, (INT, WORD)

set value
the defined ON-delay time (0 to 3276.7s)

Output

**T** (BOOL)

timer contact
is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

**EV** (INT, WORD)

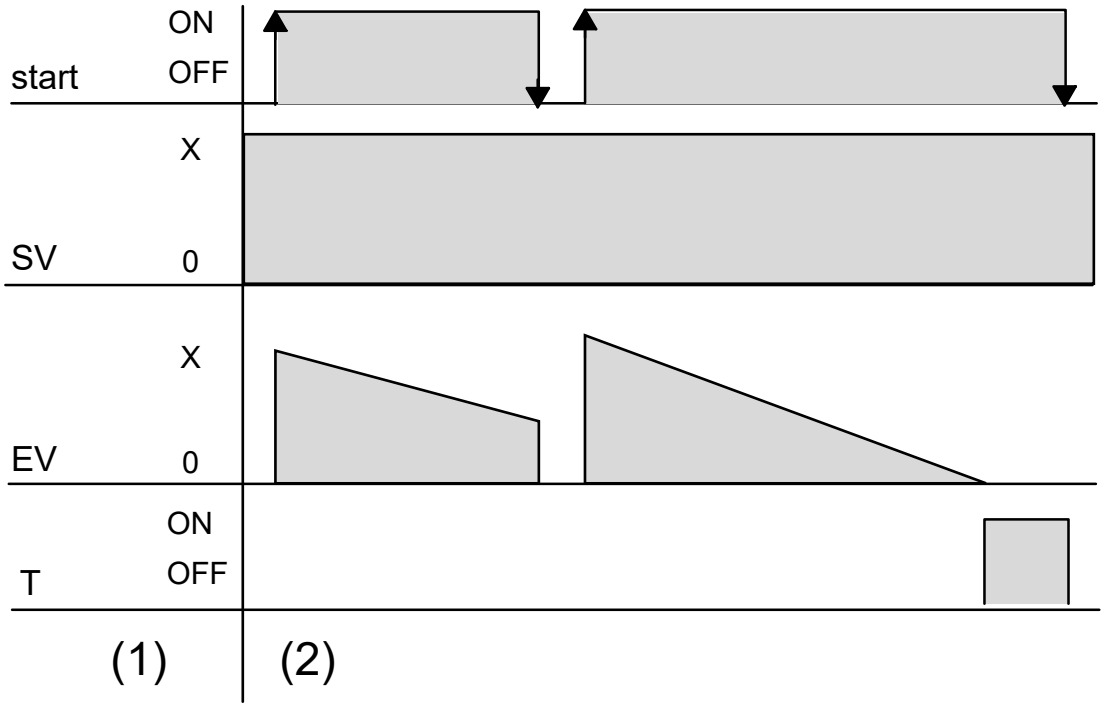
elapsed value
count value from which 1 is subtracted every 0.1s while the timer is running
The value can be changed during counting operation by writing to the variable from the programming editor.

## Remarks

- The number of available timers is limited and depends on the settings in the system registers 5 and 6.

- For the timer function blocks the compiler automatically assigns a **NUM*** address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address.

### Time chart




1. download PROG mode
2. RUN mode

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Alarm_Control | TM_100ms_FB | |
| 1 | VAR | Start_Contact | BOOL | FALSE |
| 2 | VAR | Alarm_Relay_1 | BOOL | FALSE |
| 3 | VAR | Alarm_Relay_2 | BOOL | FALSE |

This example uses variables. You may also use constants for the input variables.

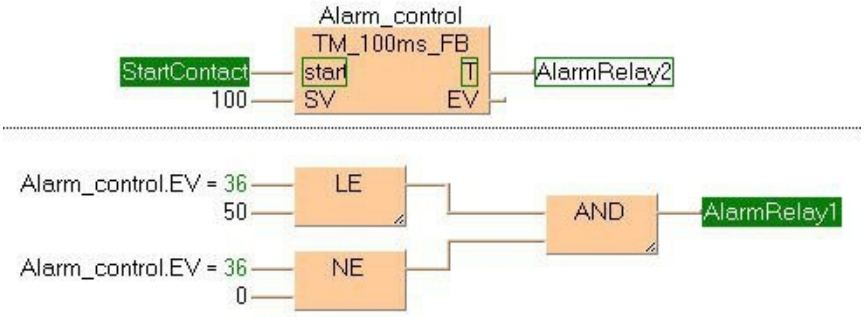
### POU body

As soon the variable **Start_contact** becomes TRUE, the timer **Alarm_control** will be started. The variable **EV** of the timer is set to the value of **SV**. As long as **Start_contact** is TRUE, the value 1 is subtracted from **EV** every 100ms. When **EV** reaches the value 0 (after

10 seconds as **SV** = 100 with the timer type TM_100ms_FB), the variable **Alarm_Relay_2** becomes TRUE.

As soon as the value of the variable **EV** of the timer is smaller than or equal to 50 (after 5s) and **EV** is unequal 0, **Alarm_Relay_1** is set to TRUE.

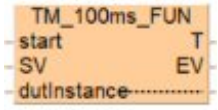
## LD body



# TM_100ms_FUN

Timer for 100ms intervals (0 to 3276.7s)

This is a user-defined function from a system function block. This timer for 0.1s units works as an ON-delay timer. If the **start** contact of the function is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.




## Parameters

Input

**start** (BOOL)

start contact
each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

**SV**, (INT, WORD)

set value
the defined ON-delay time (0 to 3276.7s)

Input/output

**dutInstance** (TM_100ms_FUN_INSTANCE_DUT)

Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

Output

**T** (BOOL)

timer contact
is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

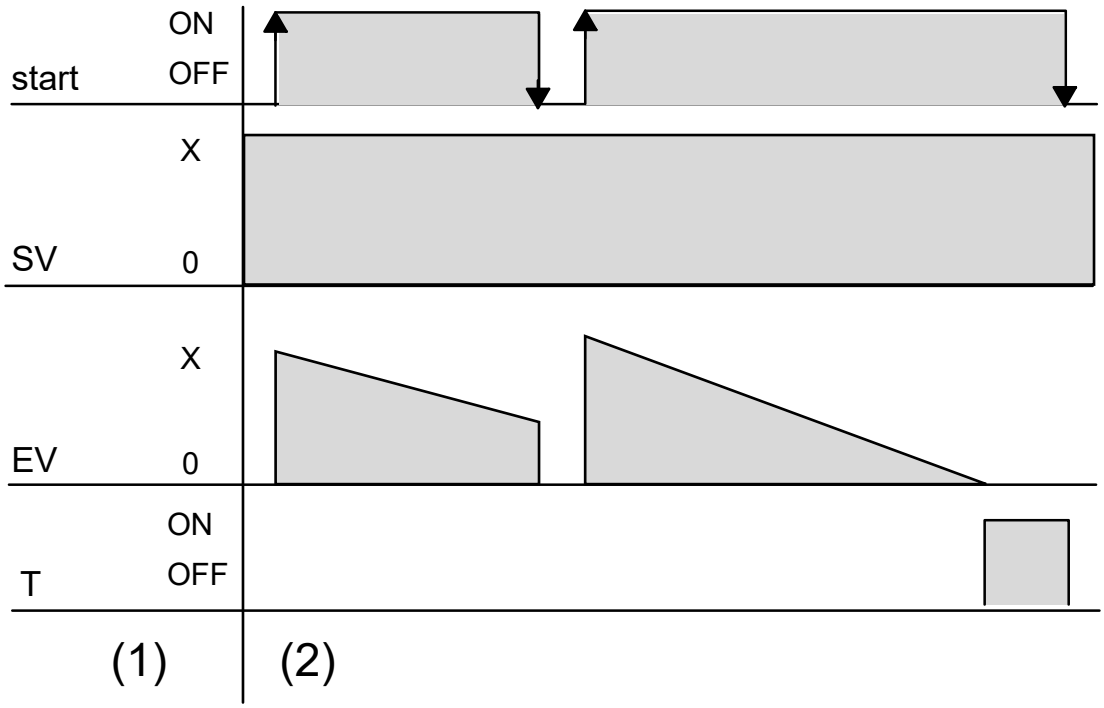
**EV** (INT, WORD)

elapsed value
count value from which 1 is subtracted every 0.1s while the timer is running
The value can be changed during counting operation by writing to the variable from the programming editor.

### Remarks

- The number of available timers is limited and depends on the settings in the system registers 5 and 6.

### Time chart

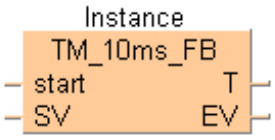



1. download PROG mode
2. RUN mode

# TM_10ms_FB

Timer for 10ms intervals (0 to 327.67s)

This timer for 0.01s units works as an ON-delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.




## Parameters

Input

**start** (BOOL)

start contact
each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

**SV** (INT, WORD)

set value
the defined ON-delay time (0 to 327.67s)

Output

**T** (BOOL)

timer contact
is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

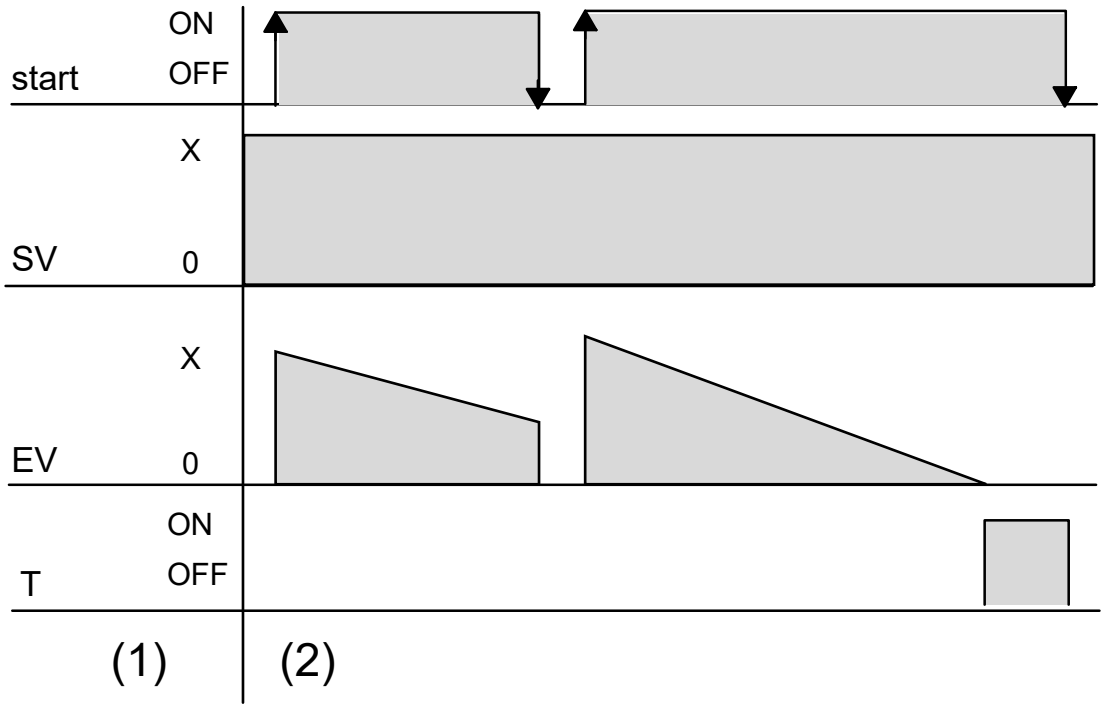
**EV** (INT, WORD)

elapsed value
count value from which 1 is subtracted every 0.01s while the timer is running
The value can be changed during counting operation by writing to the variable from the programming editor.

## Remarks

- The number of available timers is limited and depends on the settings in the system registers 5 and 6.
- For the timer function blocks the compiler automatically assigns a **NUM*** address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address.

### Time chart




(1) download PROG mode
(2) RUN mode

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Alarm_control | TM_10ms_FB | |
| 1 | VAR | Start_contact | BOOL | FALSE |
| 2 | VAR | Alarm_Relay_1 | BOOL | FALSE |
| 3 | VAR | Alarm_Relay_2 | BOOL | FALSE |

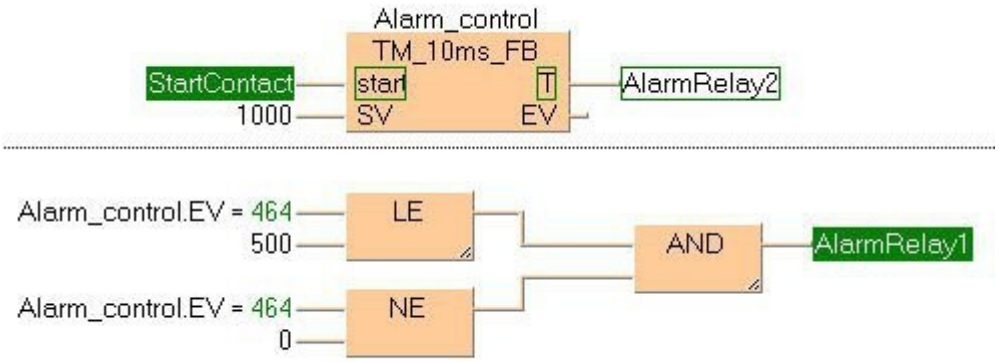
This example uses variables. You may also use constants for the input variables.

### POU body

As soon the variable **Start_contact** becomes TRUE, the timer **Alarm_control** will be started. The variable **EV** of the timer is set to the value of SV. As long as **Start_contact** is TRUE, the value 1 is subtracted from **EV** every 10ms. When **EV** reaches the value 0 (after 10 second as SV = 1000 with the timer type **TM_10ms_FB**), the variable **Alarm_Relay_2** becomes TRUE.

As soon as the value of the variable **EV** of the timer is smaller than or equal to 500 (after 5s) and **EV** is unequal 0, **Alarm_Relay_1** is set to TRUE.

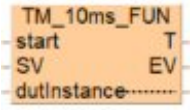
**LD body**


Alarm_control
TM_10ms_FB
StartContact
start
T
AlarmRelay2
1000
SV
EV
Alarm_control.EV = 464
500
LE
AND
AlarmRelay1
Alarm_control.EV = 464
0
NE

# TM_10ms_FUN

Timer for 10ms intervals (0 to 327.67s)

This is a user-defined function from a system function block. This timer for 0.01s units works as an ON-delay timer. If the **start** contact of the function is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.




## Parameters

Input

**start** (BOOL)

start contact
each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

**SV** (INT, WORD)

set value
the defined ON-delay time (0 to 327.67s)

Input/output

**dutInstance** (TM_10ms_FUN_INSTANCE_DUT)

Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

Output

**T** (BOOL)

timer contact
is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

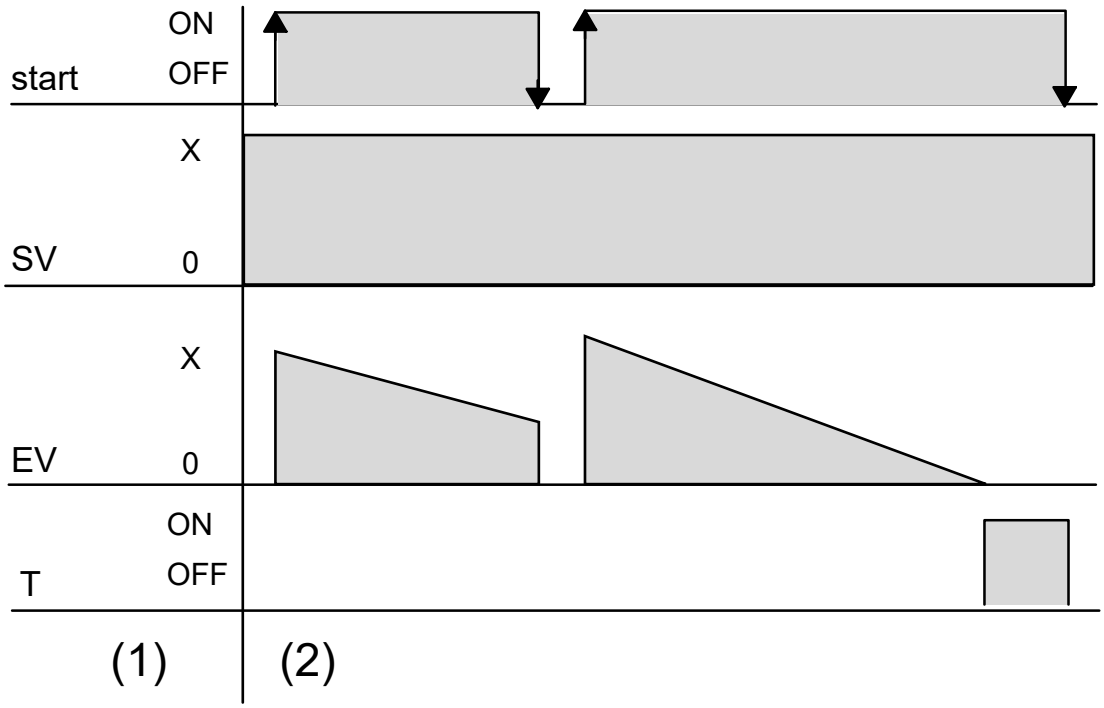
**EV** (INT, WORD)

elapsed value
count value from which 1 is subtracted every 0.01s while the timer is running
The value can be changed during counting operation by writing to the variable from the programming editor.

### Remarks

- The number of available timers is limited and depends on the settings in the system registers 5 and 6.

### Time chart

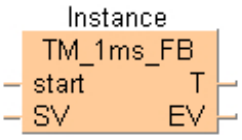



(1) download PROG mode
(2) RUN mode

# TM_1ms_FB

Timer for 1ms intervals (0 to 32.767s)

This timer for 0.001s units works as an ON-delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.




## Parameters

Input

**start** (BOOL)

start contact
each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

**SV** (INT, WORD)

set value
the defined ON-delay time (0 to 32.767s)

Output

**T** (BOOL)

timer contact
is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

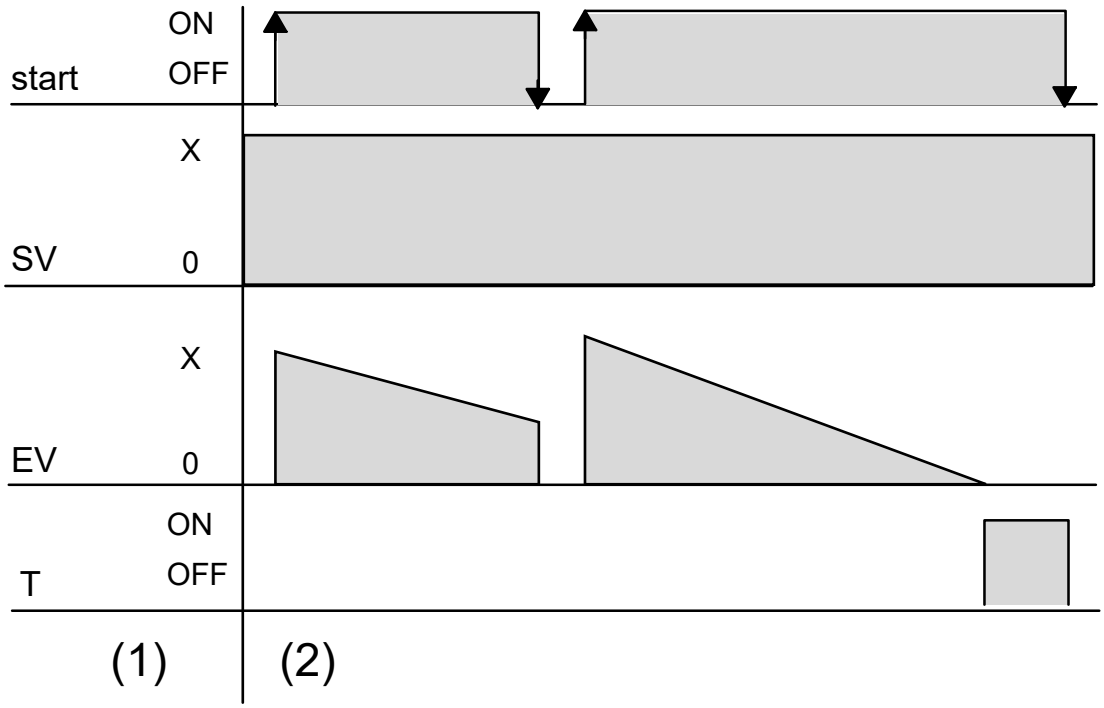
**EV** (INT, WORD)

elapsed value
count value from which 1 is subtracted every 0.001s while the timer is running
The value can be changed during counting operation by writing to the variable from the programming editor.

## Remarks

- The number of available timers is limited and depends on the settings in the system registers 5 and 6.
- For the timer function blocks the compiler automatically assigns a **NUM*** address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address.

### Time chart




(1) download PROG mode
(2) RUN mode

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Alarm_control | TM_1ms_FB | |
| 1 | VAR | Start_contact | BOOL | FALSE |
| 2 | VAR | Alarm_Relay_1 | BOOL | FALSE |
| 3 | VAR | Alarm_Relay_2 | BOOL | FALSE |

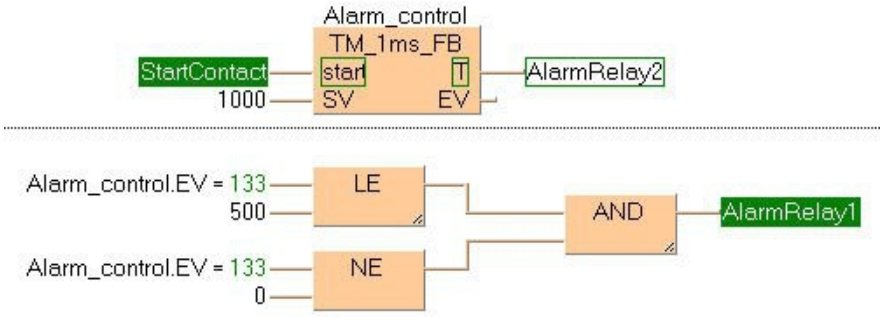
This example uses variables. You may also use constants for the input variables.

### POU body

As soon the variable **Start_contact** becomes TRUE, the timer **Alarm_control** will be started. The variable **EV** of the timer is set to the value of **SV**. As long as **Start_contact** is TRUE, the value 1 is subtracted from **EV** every 1ms. When **EV** reaches the value 0 (after 1 second as SV = 1000 with the timer type TM_1ms_FB), the variable **Alarm_Relay_2** becomes TRUE.

As soon as the value of the variable **EV** of the timer is smaller than or equal to 500 (after 0.5s) and **EV** is unequal 0, **Alarm_Relay_1** is set to TRUE.

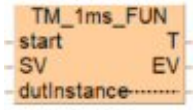
## LD body


Alarm_control
TM_1ms_FB
StartContact
start
T
AlarmRelay2
1000
SV
EV
Alarm_control.EV = 133
500
LE
AND
AlarmRelay1
Alarm_control.EV = 133
0
NE

# TM_1ms_FUN

Timer for 1ms intervals (0 to 32.767s)

This is a user-defined function from a system function block. This timer for 0.001s units works as an ON-delay timer. If the **start** contact of the function is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.




## Parameters

Input

**start** (BOOL)

start contact
each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

**SV** (INT, WORD)

set value
the defined ON-delay time (0 to 32.767s)

Input/output

**dutInstance**(TM_1ms_FUN_INSTANCE_DUT)

Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

Output

**T** (BOOL)

timer contact
is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

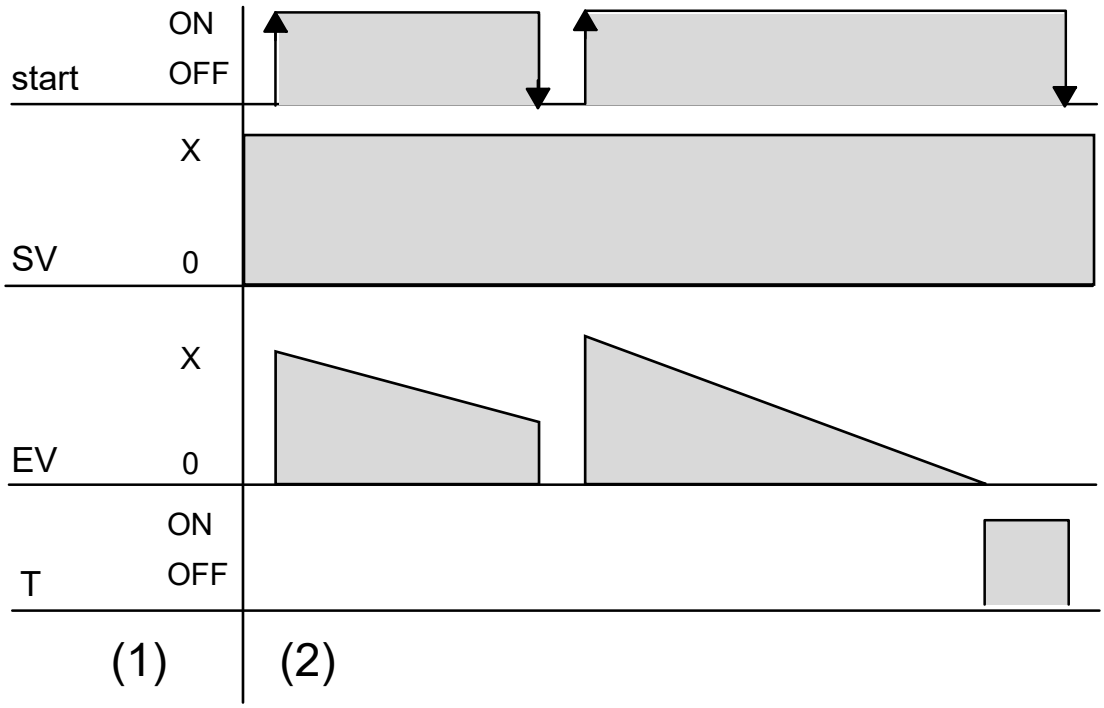
**EV** (INT, WORD)

elapsed value
count value from which 1 is subtracted every 0.001s while the timer is running
The value can be changed during counting operation by writing to the variable from the programming editor.

### Remarks

- The number of available timers is limited and depends on the settings in the system registers 5 and 6.

### Time chart

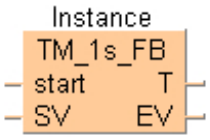



(1) download PROG mode
(2) RUN mode

# TM_1s_FB

Timer for 1s intervals (0 to 32767s)

This timer for 1s units works as an ON-delay timer. If the **start** contact of the function block is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.




## Parameters

Input

**start** (BOOL)

start contact
each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

**SV** (INT, WORD)

set value
the defined ON-delay time (0 to 32767s)

Output

**T** (BOOL)

timer contact
is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

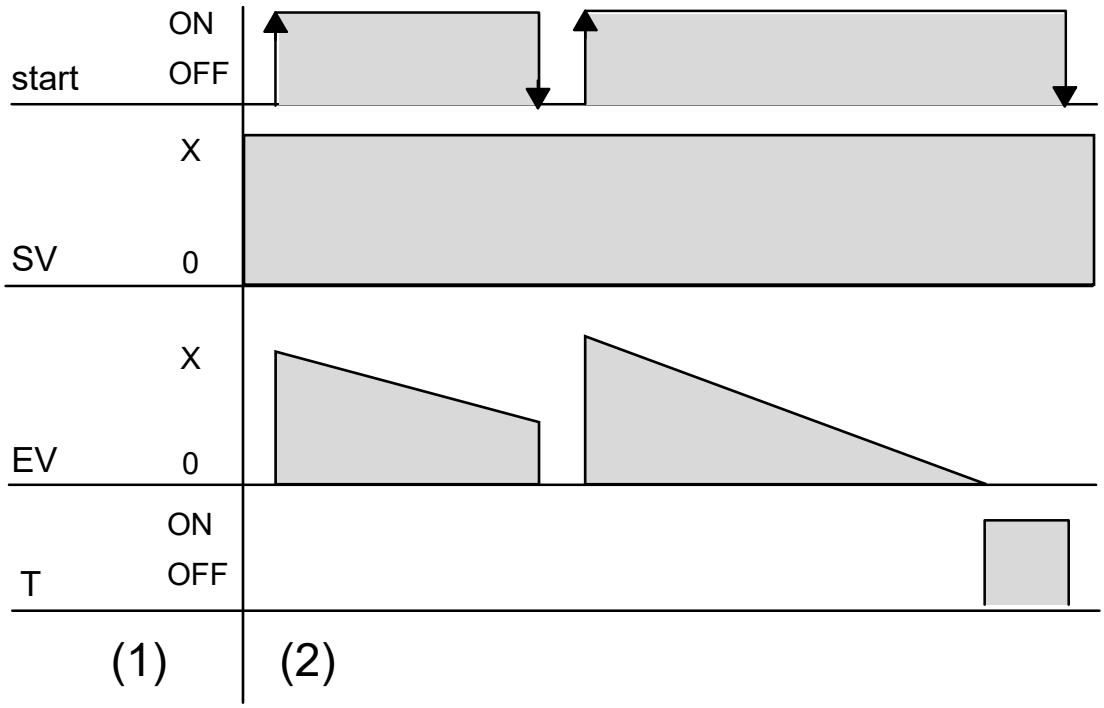
**EV** (INT, WORD)

elapsed value
count value from which 1 is subtracted every 1s while the timer is running
The value can be changed during counting operation by writing to the variable from the programming editor.

## Remarks

- The number of available timers is limited and depends on the settings in the system registers 5 and 6.
- For the timer function blocks the compiler automatically assigns a **NUM*** address to every timer instance. The addresses are assigned counting downwards, starting at the highest possible address.

### Time chart




(1) download PROG mode
(2) RUN mode

## Example

### POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | Alarm_control | TM_1s_FB | |
| 1 | VAR | Start_contact | BOOL | FALSE |
| 2 | VAR | Alarm_Relay_1 | BOOL | FALSE |
| 3 | VAR | Alarm_Relay_2 | BOOL | FALSE |

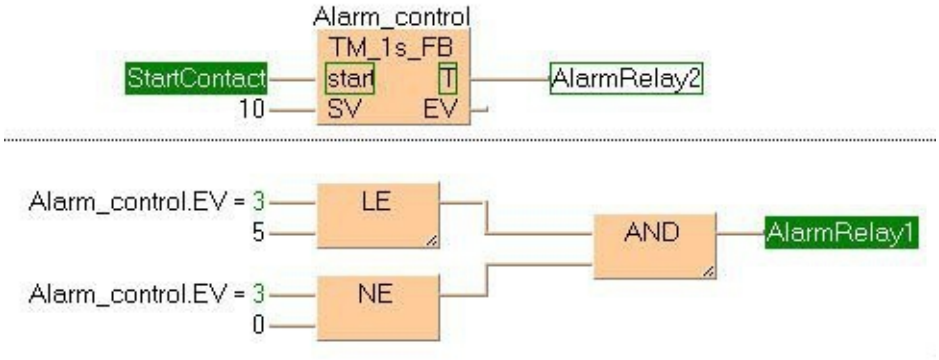
This example uses variables. You may also use constants for the input variables.

### POU body

As soon the variable **Start_contact** becomes TRUE, the timer **Alarm_control** will be started. The variable **EV** of the timer is set to the value of **SV**. As long as **Start_contact** is TRUE, the value 1 is subtracted from **EV** every 1s. When **EV** reaches the value 0 (after 10 seconds as **SV** = 10 with the timer type TM_1s_FB), the variable **Alarm_Relay_2** becomes TRUE.

As soon as the value of the variable **EV** of the timer is smaller than or equal to 5 (after 5s) and **EV** is unequal 0, **Alarm_Relay_1** is set to TRUE.

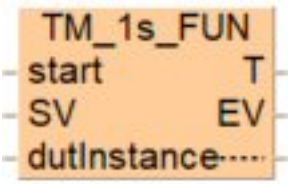
**LD body**


Alarm_control
TM_1s_FB
StartContact
start
T
AlarmRelay2
10
SV
EV
Alarm_control.EV = 3
5
LE
AND
AlarmRelay1
Alarm_control.EV = 3
0
NE

# TM_1s_FUN

Timer for 1s intervals (0 to 32767s)

This is a user-defined function from a system function block. This timer for 1s units works as an ON-delay timer. If the **start** contact of the function is in the ON state, the preset time **SV** (set value) is started. When this time has elapsed, the timer contact **T** turns ON.




## Parameters

Input

**start** (BOOL)

start contact
each time a rising edge is detected, the set value **SV** is copied to the elapsed value **EV** and the timer is started

**SV** (INT, WORD)

set value
the defined ON-delay time (0 to 32767s)

Input/output

**dutInstance** (TM_1s_FUN_INSTANCE_DUT)

Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

Output

**T** (BOOL)

timer contact
is set when the time defined at **SV** has elapsed, this means when **EV** becomes 0

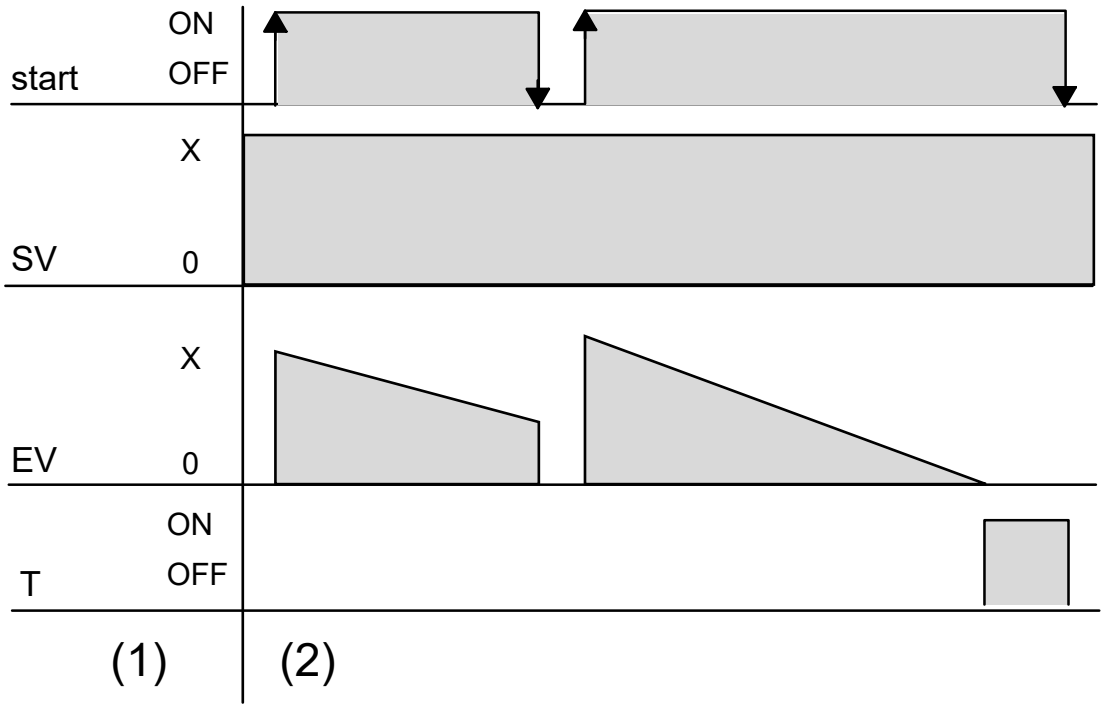
**EV** (INT, WORD)

elapsed value
count value from which 1 is subtracted every 1s while the timer is running
The value can be changed during counting operation by writing to the variable from the programming editor.

### Remarks

- The number of available timers is limited and depends on the settings in the system registers 5 and 6.

### Time chart

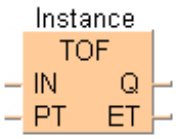



(1) download PROG mode
(2) RUN mode

# TOF

Timer with switch-off delay

The function block **TOF** allows you to program a switch-off delay, e.g. to switch off the ventilator of a machine at a later point in time than the machine itself.




## Parameters

Input

**IN** (BOOL)

timer ON
an internal timer is started if a falling edge is detected at **IN**. If a rising edge is detected at **IN** before **PT** has reached its value, **Q** will not be switched off

**PT** (TIME)

switch-off delay (PT = preset time)
16-bit value: 0–327.27s
32-bit value: 0–21,474,836.47s (32-bit value not available for FP3, FPC, FP5, FP10/10S)
resolution 10ms each

Output

**Q** (BOOL)

signal output
reset (FALSE) if **PT** = **ET**

**ET**

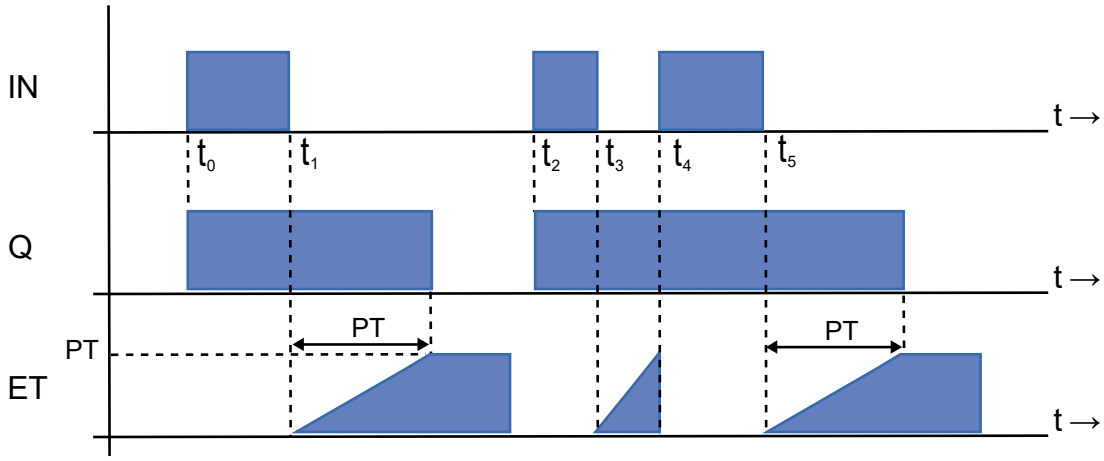
elapsed time
indicates the current value of the elapsed time
The value can be changed during counting operation by writing to the variable from the programming editor.

## Remarks

- **Q** is switched off with a delay corresponding to the time defined in **PT**. Switching on is carried out without delay.

- If **IN** (as in the time chart t3 to t4) is set prior to the lapse of the delay time **PT**, **Q** remains set (time chart for t2 to t3).

### Time chart




## Example

### POU header

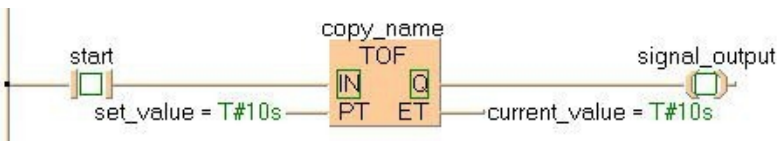
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | copy_name | TOF | |
| 1 | VAR | start | BOOL | FALSE |
| 2 | VAR | set_value | TIME | T#0s |
| 3 | VAR | signal_output | BOOL | FALSE |
| 4 | VAR | current_value | TIME | T#0s |

### POU body

If **start** is reset, this signal is transferred to **signal_output** with a delay corresponding to the period of time **set_value**.

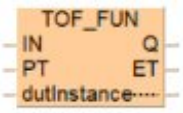
### LD body



# TOF_FUN

Timer with switch-off delay

This is a user-defined function from a system function block. **TOF_FUN** allows you to program a switch-off delay, e.g. to switch off the ventilator of a machine at a later point in time than the machine itself.




## Parameters

Input

**start** (BOOL)

timer ON
an internal timer is started if a falling edge is detected at **start**. If a rising edge is detected at **start** before **PT** has reached its value, **Q** will not be switched off

**PT** (TIME)

switch-off delay (**PT** = preset time)
16-bit value: 0–327.27s
32-bit value: 0–21,474,836.47s (32-bit value not available for FP3, FPC, FP5, FP10/10S)
resolution 10ms each

Input/output

**dutInstance** (TOF_FUN_INSTANCE_DUT)

Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

Output

**Q** (BOOL)

signal output
reset (FALSE) if **PT** = **ET**

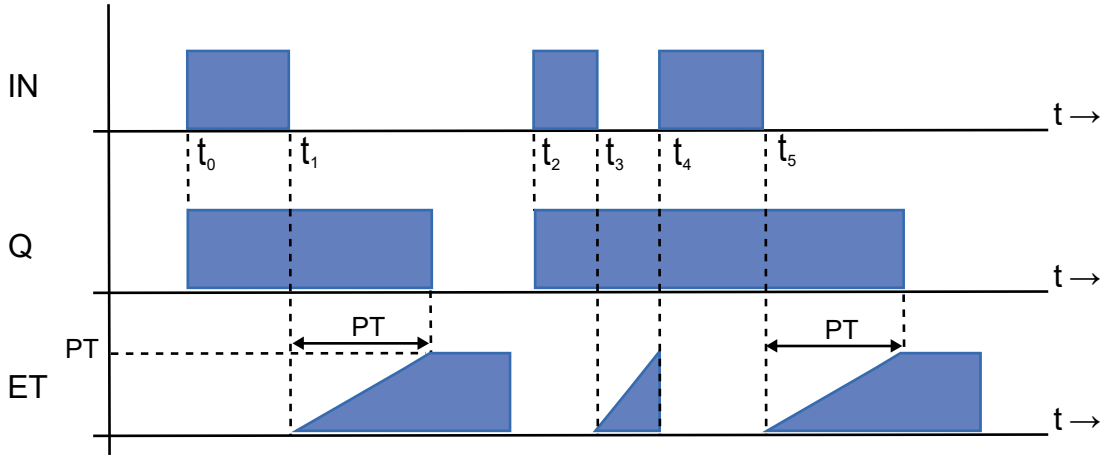
**ET**

elapsed time
indicates the current value of the elapsed time
The value can be changed during counting operation by writing to the variable from the programming editor.

### Remarks

- **Q** is switched off with a delay corresponding to the time defined in **PT**. Switching on is carried out without delay.
- If **start** (as in the time chart t3 to t4) is set prior to the lapse of the delay time **PT**, **Q** remains set (time chart for t2 to t3).

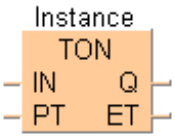
### Time chart



# TON

Timer with switch-on delay

The function block **TON** allows you to program a switch-on delay.




## Parameters

Input

**IN** (BOOL)

timer ON
an internal timer is started for each rising edge detected at **IN**

**PT** (TIME)

switch-on delay (**PT** = preset time)
16-bit value: 0–327.27s
32-bit value: 0–21,474,836.47s (32-bit value not available for FP3, FPC, FP5, FP10/10S)
resolution 10ms each

Output

**Q** (BOOL)

signal output
set to TRUE if **PT** = **ET**

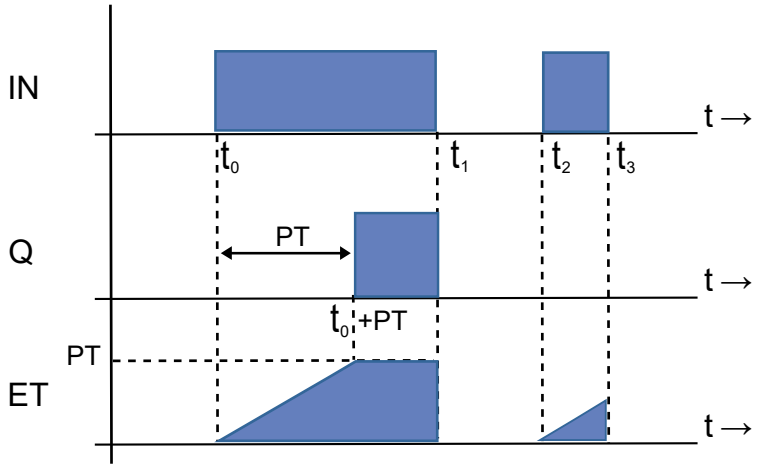
**ET** (TIME)

elapsed time
indicates the current value of the elapsed time
The value can be changed during counting operation by writing to the variable from the programming editor.

## Remarks

- **Q** is set delayed with the time defined in **PT**. Resetting is without any delay.
- If the input **IN** is only set for the period of the delay time **PT** or even for a shorter period of time (t3 - t2 < **PT**), **Q** will not be set.

### Time chart




## Example

### POU header

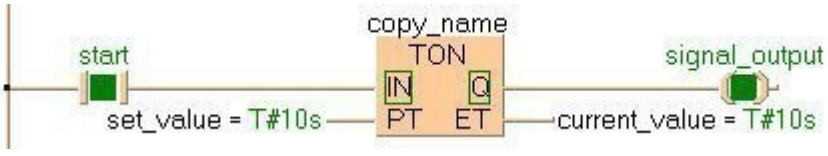
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | copy_name | TON | |
| 1 | VAR | start | BOOL | FALSE |
| 2 | VAR | set_value | TIME | T#0s |
| 3 | VAR | signal_output | BOOL | FALSE |
| 4 | VAR | current_value | TIME | T#0s |

### POU body

If **start** is set (status = TRUE), the input signal is transferred to **signal_output** with a delay by the time period **set_value**.

### LD body



## Example with emergency stop and initializing the preset time

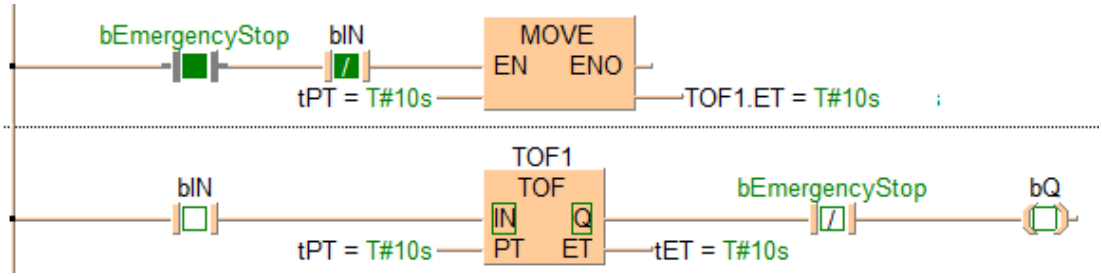
### POU header

All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

### POU body

After a falling edge at **bIN**, the timer starts counting and **Q** is set to TRUE. When **bEmergencyStop** is set to TRUE before the timer value has been elapsed, counting stops and the output **Q** is immediatiely reset. The timer is initialized with the preset time PT of 10s in this example.

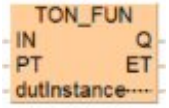
### LD body



# TON_FUN

Timer with switch-on delay

This is a user-defined function from a system function block. **TON_FUN** allows you to program a switch-on delay.




## Parameters

Input

**start** (BOOL)

timer ON
an internal timer is started for each rising edge detected at **start**

**PT** (TIME)

switch-on delay (**PT** = preset time)
16-bit value: 0–327.27s
32-bit value: 0–21,474,836.47s (32-bit value not available for FP3, FPC, FP5, FP10/10S)
resolution 10ms each

Input/output

**dutInstance** (TON_FUN_INSTANCE_DUT)

Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

Output

**Q** (BOOL)

signal output
set to TRUE if **PT** = **ET**

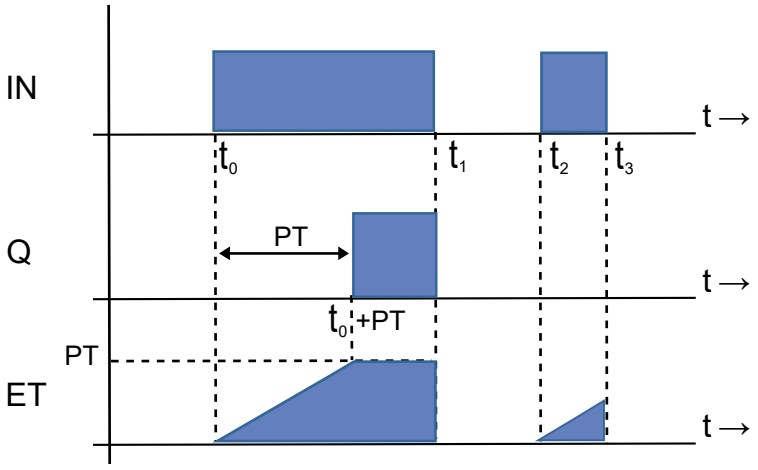
**ET** (TIME)

elapsed time
indicates the current value of the elapsed time
The value can be changed during counting operation by writing to the variable from the programming editor.

### Remarks

- **Q** is set delayed with the time defined in **PT**. Resetting is without any delay.
- If the input **start** is only set for the period of the delay time **PT** or even for a shorter period of time (t3 - t2 < **PT**), **Q** will not be set.

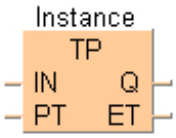
### Time chart



# TP

Timer with defined period

The function block **TP** allows you to program a pulse timer with a defined clock period.




## Parameters

Input

**IN** (BOOL)

clock generator
if a rising edge is detected at **IN**, a clock is generated having the period defined in **PT**

**PT** (TIME)

clock period
16-bit value: 0–327.27s
32-bit value: 0–21,474,836.47s (32-bit value not available for FP3, FPC, FP5, FP10/10S)
resolution 10ms each
a timer having the period **PT** is caused for each rising edge at **IN**. A new rising edge detected at **IN** within the pulse period does not cause a new timer.

Output

**Q** (BOOL)

signal output
is set for the period of **PT** as soon as a rising edge is detected at **IN**

**ET** (TIME)

elapsed time
contains the elapsed period of the timer. If **PT** = **ET**, **Q** will be reset
The value can be changed during counting operation by writing to the variable from the programming editor.

## Remarks

- FP2, FP2SH and FP10SH use a 32-bit value for **PT**.

- Independent of the turn-on period of the **IN** signal, a clock is generated at the output **Q** having a length defined by **PT**. The function block **TP** is triggered if a rising edge is detected at the input **IN**.
- A rising edge at the input **IN** does not have any influence during the processing of **PT**.

### Time chart

## Example

### POU header

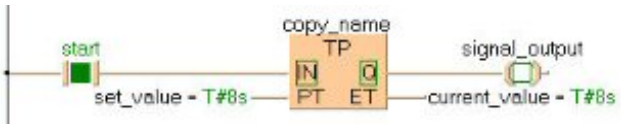
All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | copy_name | TP | |
| 1 | VAR | start | BOOL | FALSE |
| 2 | VAR | set_value | TIME | T#0s |
| 3 | VAR | signal_output | BOOL | FALSE |
| 4 | VAR | current_value | TIME | T#0s |

### POU body

If **start** is set (status = TRUE), the clock is emitted at **signal_output** until the **set_value** for the clock period is reached.

### LD body




## Example with emergency stop and initializing the preset time

### POU header

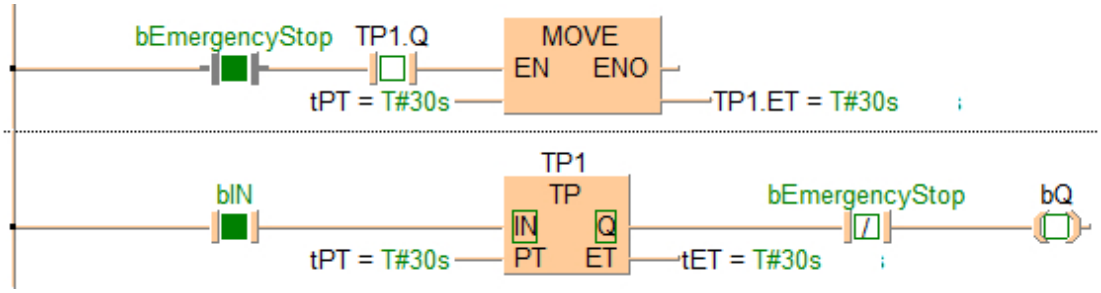
All input and output variables used for programming this function have been declared in the POU header.The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial |
|---|---|---|---|---|
| 0 | VAR | TP1 | TP | |
| 1 | VAR | bIN | BOOL | FALSE |
| 2 | VAR | tPT | TIME | T#30s |
| 3 | VAR | tET | TIME | T#0s |
| 4 | VAR | bQ | BOOL | FALSE |
| 5 | VAR | bEmergencyStop | BOOL | FALSE |

## POU body

After a rising edge at **IN**, the timer starts counting and **Q** is set to TRUE. When **bEmergencyStop** is set to TRUE before the clock period has been elapsed, counting stops and the output **Q** is immediatiely reset. The timer is initialized with the clock period PT of 30s in this example.

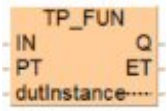
## LD body



# TP_FUN

Timer with defined period

This is a user-defined function from a system function block. **TP_FUN** allows you to program a pulse timer with a defined clock period.




## Parameters

Input

**start** (BOOL)

clock generator
if a rising edge is detected at **start**, a clock is generated having the period defined in **PT**

**PT** (TIME)

clock period
16-bit value: 0–327.27s
32-bit value: 0–21,474,836.47s (32-bit value not available for FP3, FPC, FP5, FP10/10S)
resolution 10ms each
a timer having the period **PT** is caused for each rising edge at **start**. A new rising edge detected at **start** within the pulse period does not cause a new timer.

Input/output

**dutInstance** (TP_FUN_INSTANCE_DUT)

Internal memory containing the internal values and states, which corresponds to the instance memory of the associated FB.

Output

**Q** (BOOL)

signal output
is set for the period of **PT** as soon as a rising edge is detected at **start**

**ET** (TIME)

elapsed time
contains the elapsed period of the timer. If **PT** = **ET**, **Q** will be reset
The value can be changed during counting operation by writing to the variable from the programming editor.

### Remarks

- FP2, FP2SH and FP10SH use a 32-bit value for **PT**.
- Independent of the turn-on period of the **start** signal, a clock is generated at the output **Q** having a length defined by **PT**. The function **TP_FUN** is triggered if a rising edge is detected at the input **start**.
- A rising edge at the input **start** does not have any influence during the processing of **PT**.

### Time chart

## 34.25 F instructions

Before using F instructions, we recommend trying to solve your PLC program with IEC instructions or FP instructions. IEC instructions and FP instructions have significant advantages over F instructions.

Related topics

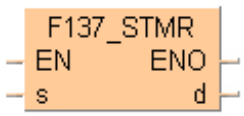
Advantages of IEC instructions

Advantages of FP instructions

# F137_STMR

Timer 16 bit

The auxiliary timer instruction **F137_STMR** is a down type timer. The formula of the timer-set time is 0.01 sec. * set value **s** (time can be set from 0.01 to 327.67 sec.). If you use the special internal flag R900D as the timer contact, be sure to program it at the address immediately after the instruction.




## Timer operation:

- If the trigger **EN** of the auxiliary timer instruction (STMR) is in the ON-state, the constant or value specified by **s** is transferred to the area specified by **d**.
- During the timing operation, the time is subtracted from the value in the area specified by **d**.
- The output ENO turns ON when the value in the area specified by **d** becomes 0.
- The variables **s** and **d** have to be of the same data type.

## Parameters

Input

**s** (WORD, INT, UINT)

16-bit area or equivalent constant for timer set value

Output

**d** (WORD, INT, UINT)

16-bit area for timer elapsed value

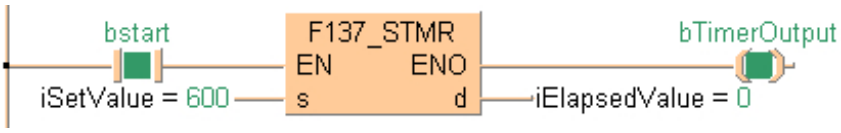
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | bstart | BOOL | FALSE | activates the timer |
| 1 | VAR | iSetValue | INT | 600 | six seconds (600 * 0,01s) |
| 2 | VAR | iElapsedValue | INT | 0 | |
| 3 | VAR | bTimerOutput | BOOL | FALSE | set to TRUE after 6s have elapsed |

## LD body



# F183_DSTM

Timer 32-bit

The F183 instruction activates an upward counting 32-bit timer which works on-delayed. The smallest counting unit is 0.01s. During execution of F183 (start = TRUE), elapsing time is added to the elapsed value **d**. The timer output will be enabled when the elapsed value **d** equals the set value **s**. If the **start** condition **EN** is set to FALSE, execution will be interrupted and the elapsed value **d** will be reset to zero. The set value **s** can be changed during execution of **F183**.


## Parameters

Input

**s** (DWORD, DINT, UDINT, DATE, TOD, DT)

set value, range 0 to 2147483647

Output

**d** (DWORD, DINT, UDINT, DATE, TOD, DT)

elapsed value, range 0 to 2147483647

## Remarks

The delay time of the timer can be calculated using the following formula: (Set Value **s**) * (0.01s) = on-delay

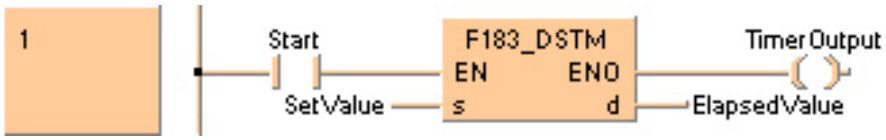
## Example

## POU header

All input and output variables used for programming this function have been declared in the POU header. The same POU header is used for all programming languages.

| | Class | Identifier | Type | Initial | Comment |
|---|---|---|---|---|---|
| 0 | VAR | Start | BOOL | FALSE | |
| 1 | VAR | SetValue | DINT | 0 | 10 seconds |
| 2 | VAR | TimerOutput | BOOL | FALSE | turns on when 10s have elapsed |
| 3 | VAR | ElapsedValue | DINT | 0 | |

### LD body


1
Start
SetValue
F183_DSTM
EN
ENO
s
d
TimerOutput
ElapsedValue

# 35 Appendix

## 35.1 FP7 instructions

| FPWIN Pro instruction | 32-bit PLC instruction (FPWIN GR) | 16-bit PLC instruction |
|---|---|---|
| EQ<br>GT<br>LT<br>NE<br>GE<br>LE | CMP | F60_CMP<br>F61_DCMP<br>F345_FCMP |
| WITHIN_LIMITS | WIN | |
| FP_COMPARE_BLOCK | BCMP | F64_BCMP |
| FP_GET_IO_START_OFFSET<br>FP_GET_UNIT_OFFSETS1 | GETSTNO | F4_GETS |
| MOVE | MV | F0_MV<br>F1_DMV |
| FP_MOVE_INVERT | MVN | F2_MVN<br>F3_DMVN |
| FP_MOVE_BLOCK | BKMV | F10_BKMV |
| FP_COPY | COPY | F11_COPY |
| FP_MOVE_BITS | BTM | F5_BTM |
| FP_MOVE_DIGITS | DGT | F6_DGT |
| FP_EXCHANGE | XCH | F15_XCH<br>F16_DXCH |
| FP_SWAP_BYTES | SWAP | F17_SWAP |
| ADD | ADD | |
| SUB | SUB | |
| MUL | MUL | |
| DIV | DIV | |
| FP_INC | INC | F35_INC<br>F36_DINC |
| FP_DEC | DEC | F37_DEC<br>F38_DDEC |
| FP_DIV_MOD | DIVMOD | F32_DIV<br>F33_DDIV |

| FPWIN Pro instruction | 32-bit PLC instruction (FPWIN GR) | 16-bit PLC instruction |
|---|---|---|
| FP_ADD_BCD | BCDADD | F40_BADD<br>F41_DBADD |
| FP_SUB_BCD | BCDSUB | F45_BSUB<br>F46_DBSUB |
| FP_MUL_BCD | BCDMUL | F50_BMUL<br>F51_DBMUL |
| FP_DIV_MOD_BCD | BCDDIV | F52_BDIV<br>F53_DBDIV |
| FP_DIV_BCD | BCDDIV | F52_BDIV<br>F53_DBDIV |
| FP_MOD_BCD | BCDDIV | F52_BDIV<br>F53_DBDIV |
| FP_INC_BCD | BCDINC | F55_BINC<br>F56_DBINC |
| FP_DEC_BCD | BCDDEC | F57_BDEC<br>F58_DBDEC |
| AND | AND | |
| OR | OR | |
| XOR | XOR | |
| FP_COMBINE | COMB | F69_WUNI<br>F219_DUNI |
| FP_BCC | BCC | F70_BCC |
| FP_CRC | CRC | CRC16<br>F70_BCC |
| FP_HEX_TO_ASCII | HEXA | F71_HEX2A |
| FP_ASCII_TO_HEX | AHEX | F72_A2HEX |
| FP_BCD_TO_ASCII | BCDA | F73_BCD2A |
| FP_ASCII_TO_BCD | ABCD | F74_A2BCD |
| FP_DEC_TO_ASCII | BINA | F75_BIN2A<br>F77_DBIN2A |
| FP_ASCII_TO_DEC | ABIN | F76_A2BIN<br>F78_DA2BIN |
| FP_ASCII_TO_BIN | ATOB | F251_ATOB |
| FP_BIN_TO_ASCII | BTOA | F250_BTOA |
| FP_CHECK_ASCII | ACHK | F252_ACHK |
| FP_INVERT | INV | F84_INV |

| FPWIN Pro instruction | 32-bit PLC instruction (FPWIN GR) | 16-bit PLC instruction |
|---|---|---|
| ABS | ABS | F87_ABS<br>F88_DABS<br>F336_FABS |
| WORD_TO_BCD_WORD | BCD | |
| BCD_WORD_TO_WORD | BIN | |
| FP_DECODE | DECO | F90_DECO |
| FP_7SEGMENT | SEGT | F91_SEGT |
| FP_ENCODE | ENCO | F92_ENCO |
| FP_UNIFY_DIGITS | UNIT | F93_UNIT |
| FP_DIVIDE_DIGITS | DIST | F94_DIST |
| FP_UNIFY_BYTES | BUNI | F69_WUNI<br>F219_DUNI |
| FP_DIVIDE_BYTES | BDIS | |
| FP_BIN_TO_GRAY | GRY | F235_GRY<br>F236_DGRY |
| FP_GRAY_TO_BIN | GBIN | F237_GBIN<br>F238_DGBIN |
| FP_WORD_TO_BITCOLUMN | COLM | F240_COLM |
| FP_BITCOLUMN_TO_WORD | LINE | F241_LINE |
| FP_SHR_BLOCK | BITR | F108_BITR |
| FP_SHL_BLOCK | BITL | F109_BITL |
| FP_WSHR_BLOCK | WSHR | F110_WSHR |
| FP_WSHL_BLOCK | WSHL | F111_WSHL |
| ROR | ROR | F120_ROR |
| ROL | ROL | F121_ROL |
| FP_ROR_CARRY | RCR | F122_RCR |
| FP_ROL_CARRY | RCL | F123_RCL |
| FP_DATA_READ_COMPRESS | CMPR | F98_CMPR |
| FP_DATA_WRITE_COMPRESS | CMPW | F99_CMPW |
| FP_FIFO_DEFINE | DEFBUF | F115_FIFT |
| FP_LIFO_DEFINE | DEFBUF | F115_FIFT |
| FP_FIFO_READ | FIFR | F116_FIFR |
| FP_FIFO_WRITE | FIFW | F117_FIFW |
| FP_LIFO_WRITE | FIFW | F117_FIFW |
| FP_LIFO_READ | LIFR | F116_FIFR |

| FPWIN Pro instruction | 32-bit PLC instruction (FPWIN GR) | 16-bit PLC instruction |
|---|---|---|
| FP_INVERT_BIT | BTI | F132_BTI |
| FP_TEST_BIT | BTT | F133_BTT |
| EQ<br>GT<br>LT<br>NE<br>GE<br>LE | ESCMP | |
| CONCAT | ESADD | |
| LEN | ELEN | |
| FIND | ESSRC | |
| RIGHT | ERIGHT | |
| LEFT | ELEFT | |
| MID | EMIDR | |
| REPLACE<br>INSERT<br>DELETE | ESREP | |
| FP_DATA_SEARCH | SRC | F96_SRC |
| FP_COUNT_TRUE_BITS | BCU | F135_BCU<br>F136_DBCU |
| FP_DATA_MAX_POS | MAX | MAX<br>F270_MAX<br>F271_DMAX |
| FP_DATA_MIN_POS | MIN | MIN<br>F272_MIN<br>F273_DMIN |
| FP_DATA_MEAN_SUM_INT<br>FP_DATA_MEAN_SUM_UINT<br>FP_DATA_MEAN_SUM_REAL<br>FP_DATA_MEAN_SUM_DINT0<br>FP_DATA_MEAN_SUM_UDINT0 | MEAN | F275_MEAN<br>F276_DMEAN<br>F352_FMEAN |
| FP_DATA_SORT | SORT | F277_SORT<br>F278_DSORT<br>F353_FSORT |
| FP_SCALE | SCAL | F282_SCAL<br>F283_DSCAL<br>F354_FSCAL |
| FP_EVENTS_COUNT | EVENTC | |
| FP_EVENTS_OPERATION_TIME | EVENTT | |

| FPWIN Pro instruction | 32-bit PLC instruction (FPWIN GR) | 16-bit PLC instruction |
|---|---|---|
| FP_PID_BASIC | PID | F355_PID_DUT |
| FP_PID | EZPID | F356_PID_PWM |
| FP_DETECT_CHANGE | DTR | F373_DTR<br>F374_DDTR |
| FP_RAMP | RAMP | F284_RAMP |
| LIMIT | LIMT | |
| FP_BAND | BAND | |
| FP_ZONE | ZONE | |
| FP_DEBOUNCE | FILTR | F182_FILTER |
| SIN | SIN | F314_SIN |
| COS | COS | F315_COS |
| TAN | TAN | |
| ASIN | ASIN | |
| ACOS | ACOS | |
| ATAN | ATAN | |
| FP_ATAN2 | ATAN2 | ATAN2_YX |
| FP_SINH | SINH | |
| FP_COSH | COSH | |
| FP_TANH | TANH | |
| EXP | EXP | |
| LN | LN | |
| LOG | LOG | |
| EXPT | PWR | |
| SQRT | SQR | |
| FP_RAD | RAD | F337_RAD |
| FP_DEG | DEG | F338_DEG |
| FP_ROUND_DOWN | FINT | F333_FINT |
| FP_ROUND | FRINT | F334_FRINT |
| ABS | FABS | |
| INT_TO_REAL<br>DINT_TO_REAL | FLT | |
| REAL_TO_INT<br>REAL_TO_DINT | INT | |
| TRUNC_TO_INT<br>TRUNC_TO_DINT | FIX | |

| FPWIN Pro instruction | 32-bit PLC instruction (FPWIN GR) | 16-bit PLC instruction |
|---|---|---|
| FP_DTBIN_TO_SEC | TMSEC | |
| FP_SET_ERROR | ERR | F148_ERR |
| TON_FB | SPTM | |
| FP_RESET_WATCHDOG | WDTRES | F142_WDT |
| FP_LOGTRACE_SAMPLE | SMPL | |
| FP_LOGTRACE_START | LOGST | |
| FP_LOGTRACE_STOP | LOGED | |
| SEND_DATA<br>SendCharacters | GPSEND | |
| ReceiveData<br>ReceiveCharacters | GPRECV | |
| FP_WRITE_TO_SLAVE<br>FP_WRITE_TO_SLAVE_AREA_OFFS<br>FP_MODBUS_MASTER | SEND | |
| FP_READ_FROM_SLAVE<br>FP_READ_FROM_SLAVE_AREA_OFFS<br>FP_MODBUS_MASTER | RECV | |
| FP_ETHERNET_GET_STATUS | RDET | |
| FP_ETHERNET_PING | PINGREQ | |
| FP_COM_GET_STATUS | PMGET | |
| FP_COM_GET_PARAMETER | PMGET | |
| FP_COM_SET_PARAMETER | PMSET | |
| FP_ETHERNET_CONNECTION_SET | CONSET | — |
| FP_ETHERNET_CONNECTION_OPEN | OPEN | — |
| FP_ETHERNET_CONNECTION_CLOSE | CLOSE | — |
| FP_IPV4_SET_ADDRESS | IPv4SET | — |
| FP_IPV4_GET_CONNECTION | ETSTAT | — |
| FP_IPV4_GET_MAC | ETSTAT | — |
| FP_IPV6_GET_CONNECTION | ETSTAT | — |
| FP_IPV6_GET_MAC | ETSTAT | — |
| FP_FTP_GET_STATUS | ETSTAT | — |
| FP_FTP_GET_STATUS_ALL | ETSTAT | — |
| FP_SMTP_GET_STATUS | ETSTAT | — |
| FP_SMTP_GET_STATUS_ALL | ETSTAT | — |
| FP_HTTP_GET_STATUS | ETSTAT | — |
| FP_HTTP_GET_STATUS_ALL | ETSTAT | — |

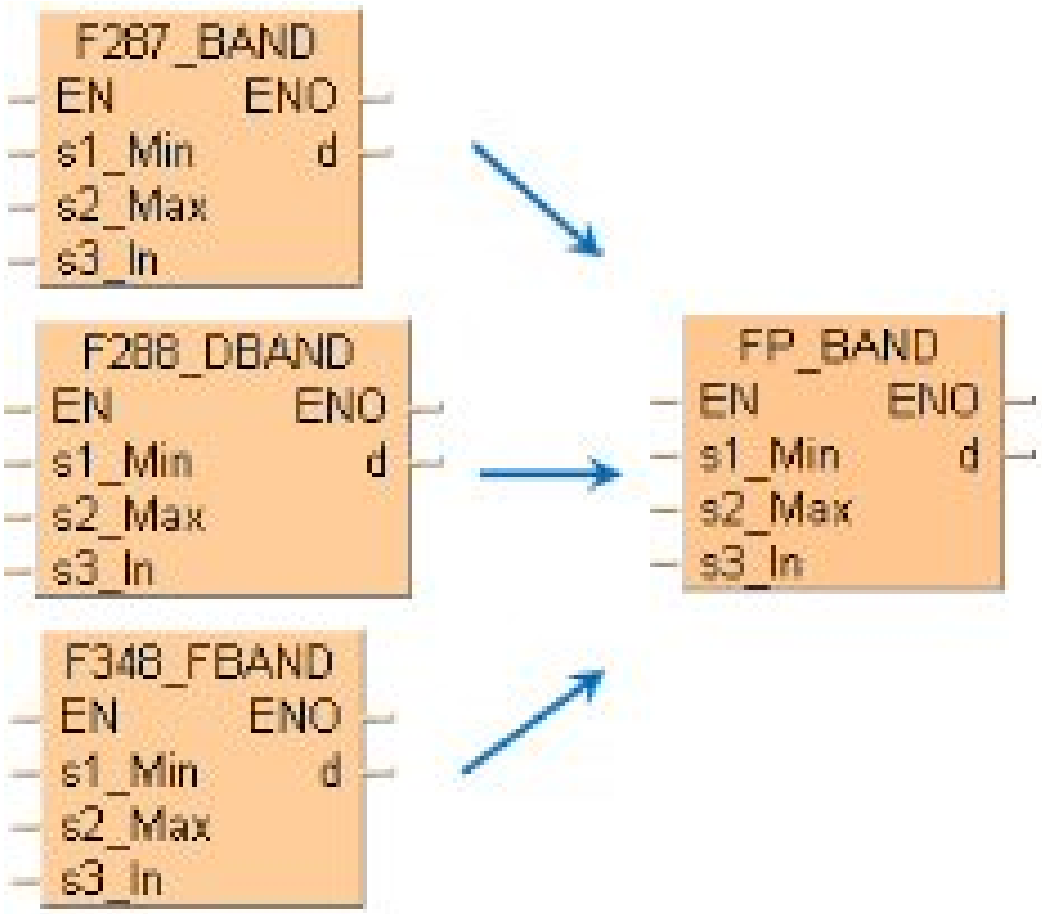
| FPWIN Pro instruction | 32-bit PLC instruction (FPWIN GR) | 16-bit PLC instruction |
|---|---|---|
| FP_FTP_SET_CONNECTION | FTPcSV | — |
| FP_FTP_SET_MODE | FTPcSET | — |
| FP_FTP_TRANSFER_REQUEST | FTPcREQ | — |
| FP_FTP_TRANSFER_CONTROL | FTPcCTL | — |
| FP_FTP_TRANSFER_CONTROL_ALL | FTPcCTL | — |
| FP_HTTP_SET_CONNECTION | HTTPcSV | — |
| FP_HTTP_SET_MODE | HTTPcSET | — |
| FP_HTTP_TRANSFER_REQUEST | HTTPcREQ | — |
| FP_HTTP_TRANSFER_CONTROL | HTTPcCTL | — |
| FP_HTTP_TRANSFER_CONTROL_ALL | HTTPcCTL | — |
| FP_SMTP_GET_EMAIL_TEXT | SMTPcBDY | |
| FP_SMTP_SET_EMAIL_TEXT | SMTPcBRD | |
| FP_SMTP_SET_MODE | SMTPcSET | — |
| FP_SMTP_SET_GROUP | SMTPcADD | — |
| FP_SMTP_SET_CONNECTION | SMTPcSV | — |
| FP_SMTP_TRANSFER_REQUEST | SMTPcREQ | — |
| FP_SD_WRITE_BIN | CDTWT | — |
| FP_SD_READ_BIN | CDTRD | — |
| FP_SD_WRITE | CWT | — |
| FP_SD_READ | CRD | — |
| FP_SD_CREATE_DIR | CMKDIR | — |
| FP_SD_DELETE_DIR | CRMDIR | — |
| FP_SD_DELETE_FILE | CFDEL | — |
| FP_SD_WRITE_LINE | CPR | — |
| FP_SD_READ_LINE | CRD1 | — |
| FP_SD_COPY_FILE | CCOPY | — |
| FP_SD_MOVE_FILE | CMV | — |
| FP_SD_RENAME_FILE | CREN | — |
| FP_SD_GET_FREE_KBYTES | CFREEK | — |
| FP_SD_GET_FILE_STATUS | CFLS | — |
| FP_POS_UNIT_SET_TABLE | POSSET | |
| FP_POS_UNIT_GET_STATUS | PSTRD | |
| FP_POS_UNIT_GET_ERROR | PERRD | |
| FP_CLEAR_UNIT_ERROR | UCLR | |

| FPWIN Pro instruction | 32-bit PLC instruction (FPWIN GR) | 16-bit PLC instruction |
|---|---|---|
| FP_INTERRUPT_ENABLE | EI | |
| FP_INTERRUPT_DISABLE | DI | |
| FP_INTERRUPT_ACTIVATE | IMASK | |
| FP_INTERRUPT_CLEAR_REQUESTS | ICLR | |
| FP_END_SCAN (page 1488) | CNDE | |

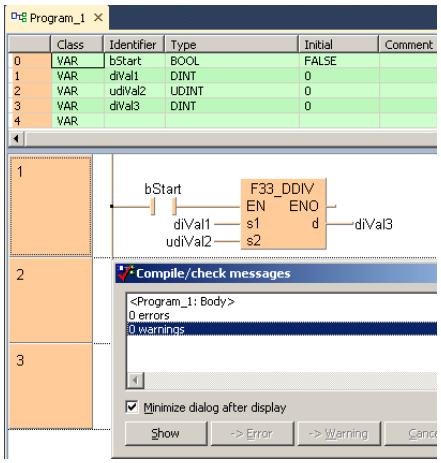
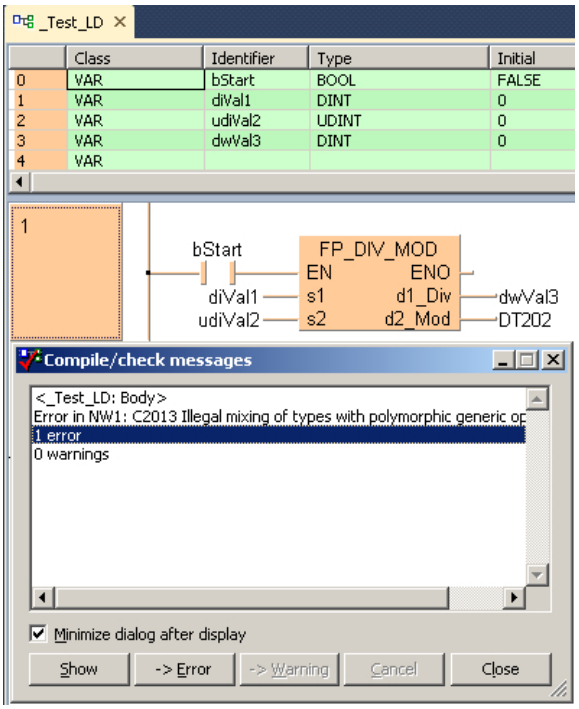
## 35.2 Advantages of FP instructions

FP instructions start with the prefix FP_. They can be selected from the FP library. When selecting instructions by category, look for the subsection "FP instructions". FP instructions have the following advantages over F instructions:

- Overloaded instructions reduce the number of instructions with which you need to be familiar.




- Data type safe: the compiler reserves the memory for input variables as well as for output variables.

| Using an F instruction, e.g. F33_DDIV | Using an overloaded FP instruction, e.g. FP_DIV_MOD |
|---|---|
|  |  |
| No error message is issued. | An error message is issued. |



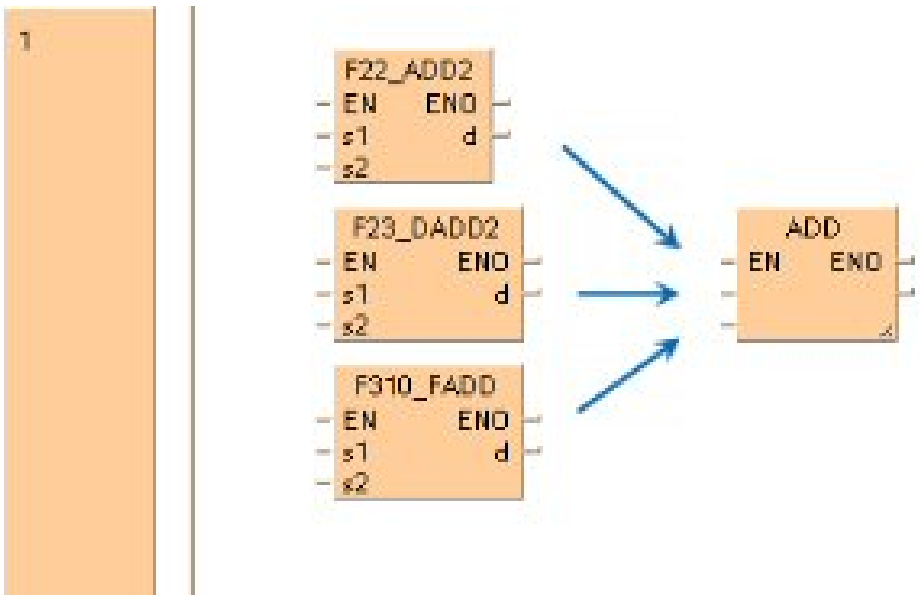
- Overloaded instructions are also available for PLCs processing 16-bit data types only.

Related topics

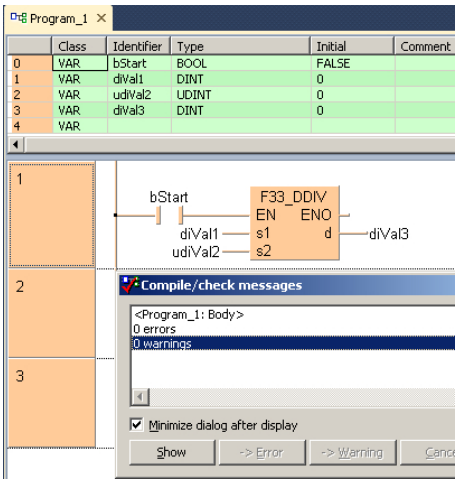
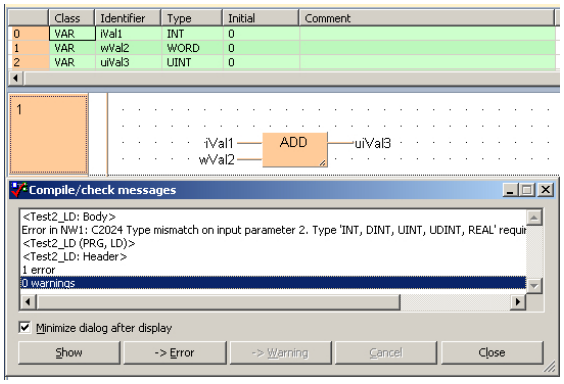
Advantages of IEC instructions (page 1949)

## 35.3 Advantages of IEC instructions

- Reduce number of instructions with which you need to be familiar.

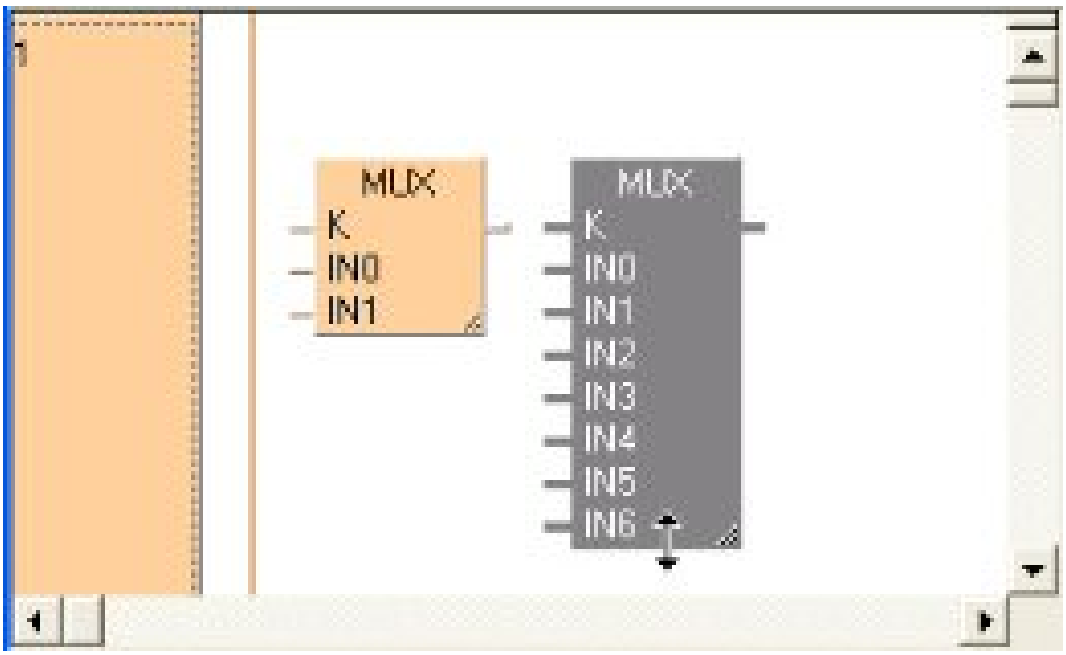



- Data type safe

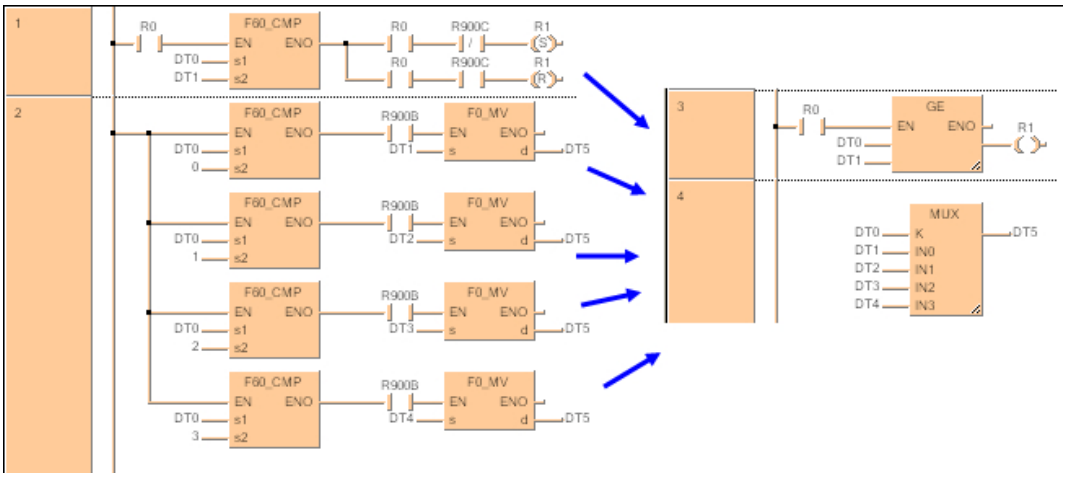
| Using FP instruction, e.g. **F22_ADD2** | Using IEC instruction **ADD** |
|---|---|
|  |  |
| You receive no error message | You receive an error message |



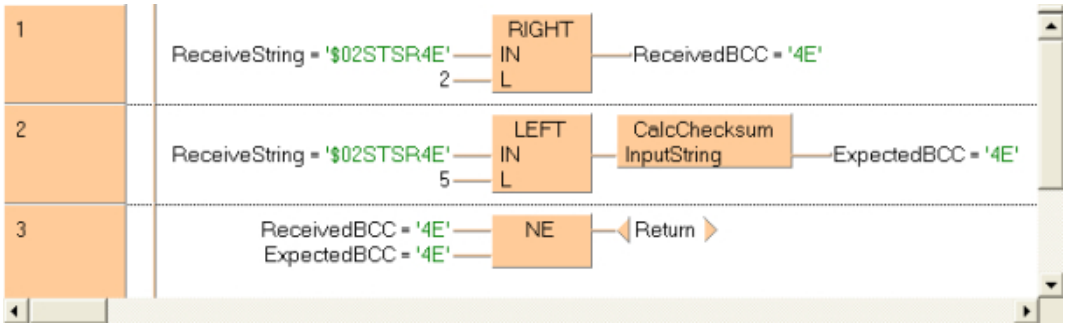
- IEC instructions encapsulate complex functionality
- IEC instructions are extensible.




- IEC instructions are predefined.




- Less programming effort
- Clear programming
- Easy to read
- All STRING instructions are available for all PLCs.



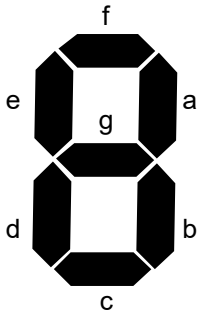

- All STRING instructions can be used even if the PLC does not support the data type STRING.

## 35.4 7-Segment conversion table

Organization of 7-segment indication

| a | b | c | d | e | f | g | |
|---|---|---|---|---|---|---|---|

LSB MSB




| One digit data to be converted | 8-bit data for 7-segment indication | | | | | | | | 7-segment indication |
|---|---|---|---|---|---|---|---|---|---|
| | - | g | f | e | d | c | b | a | |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | |
| 2 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | |
| 3 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | |
| 4 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | |
| 5 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | |
| 6 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | |
| 7 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | |
| 8 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |

| One digit data to be converted | 8-bit data for 7-segment indication | | | | | | | | 7-segment indication |
|---|---|---|---|---|---|---|---|---|---|
| 9 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |  |
| A | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |  |
| B | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |  |
| C | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |  |
| D | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |  |
| E | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |  |
| F | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |  |

## 35.5 BCD data

BCD is an acronym for **b**inary-**c**oded **d**ecimal, and means that each digit of a decimal number is expressed as a binary number.

| Decimal number | 6 | 4 | 5 |
|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ |
| BCD (binary-coded decimal) | 0110 | 0100 | 0101 |

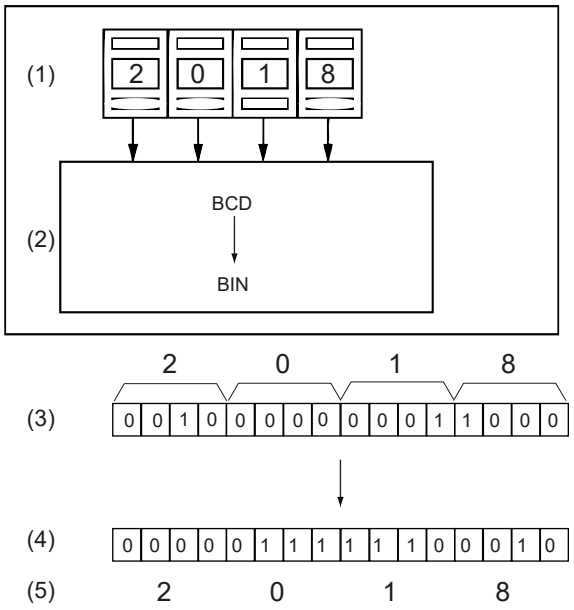
### Handling BCD data in the PLC

When inputting data from a digital switch to the PLC or outputting data to a 7-segment display (with a decoder), the data must be in BCD format. In such cases, use a data conversion instruction as shown in the examples at below.

BCD arithmetic instructions (F40 to F58) allow BCD data to be used directly. However, since PLCs compute in binary format, BIN operation instructions (F20 to F38) are more convenient.

Input from a digital switch

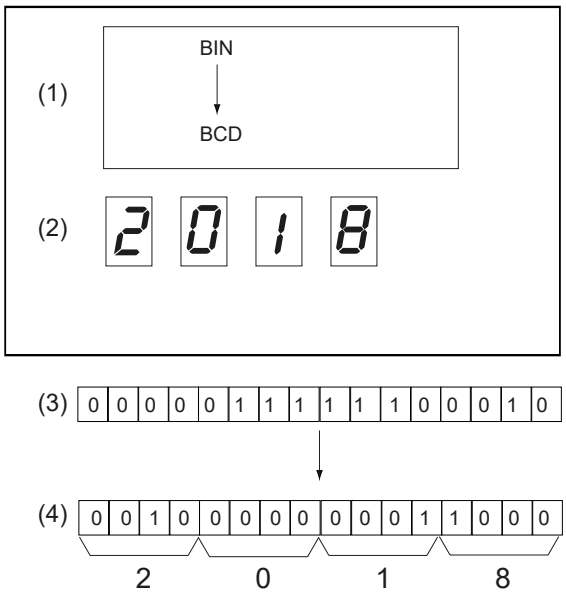
Use the BCD to BIN conversion instruction **F81_BIN**.




(1) Digital switch
(2) PLC using **F81_BIN**
(3) BCD data input
(4) BIN data that can be processed in the PLC
(5) converted to 2018 in decimal data

Output to a 7-segment display (with decoder)

Use the BIN to BCD conversion instruction **F80_BCD**.




(1) PLC using **F80_BCD**
(2) BIN data that can be processed in the PLC
(3) BCD data output

Related topics

F81_BIN

F80_BCD

## 35.6 Decimal to binary/BCD/gray code table

- Decimal number e.g. 01234567

  Binary data (1 byte):
  0000 0000 0000 0000 0000 0000 0000 0001
  0000 0000 0000 0010 0000 0000 0000 0011
  0000 0000 0000 0100 0000 0000 0000 0101
  0000 0000 0000 0110 0000 0000 0000 0111

  BCD data:
  0000 0000 0000 0000 0000 0000 0000 0001
  0000 0000 0000 0010 0000 0000 0000 0011
  0000 0000 0000 0100 0000 0000 0000 0101
  0000 0000 0000 0110 0000 0000 0000 0111

  Gray code:
  0000 0000 0000 0000 0000 0000 0000 0001
  0000 0000 0000 0011 0000 0000 0000 0010
  0000 0000 0000 0110 0000 0000 0000 0111
  0000 0000 0000 0101 0000 0000 0000 0100

- Decimal number e.g. 89101112131415

  Binary data (1 byte):
  0000 0000 0000 1000 0000 0000 0000 1001
  0000 0000 0000 1010 0000 0000 0000 1011
  0000 0000 0000 1100 0000 0000 0000 1101
  0000 0000 0000 1110 0000 0000 0000 1111

  BCD data:
  0000 0000 0000 1000 0000 0000 0000 1001
  0000 0000 0001 0000 0000 0000 0001 0001
  0000 0000 0001 0010 0000 0000 0001 0011
  0000 0000 0001 0100 0000 0000 0001 0101

  Gray code:
  0000 0000 0000 1100 0000 0000 0000 1101
  0000 0000 0000 1111 0000 0000 0000 1110
  0000 0000 0000 1010 0000 0000 0000 1011
  0000 0000 0000 1001 0000 0000 0000 1000

- Decimal number e.g. 1617181920212223

  Binary data (1 byte):
  0000 0000 0001 0000 0000 0000 0001 0001
  0000 0000 0001 0010 0000 0000 0001 0011
  0000 0000 0001 0100 0000 0000 0001 0101
  0000 0000 0001 0110 0000 0000 0001 0111

  BCD data:
  0000 0000 0001 0110 0000 0000 0001 0111
  0000 0000 0001 1000 0000 0000 0001 1001
  0000 0000 0010 0000 0000 0000 0010 0001
  0000 0000 0010 0010 0000 0000 0010 0011

  Gray code:

0000 0000 0001 1000 0000 0000 0001 1001
0000 0000 0001 1011 0000 0000 0001 1010
0000 0000 0001 1110 0000 0000 0001 1111
0000 0000 0001 1101 0000 0000 0001 1100

- Decimal number e.g.

  32...63

  Binary data (1 byte):
  0000 0000 0010 0000...0000 0000 0011 1111

  BCD data:
  0000 0000 0011 0010...0000 0000 0110 0011

  Gray code:
  0000 0000 0011 0000...0000 0000 0010 0000

- Decimal number e.g.

  64...255

  Binary data (1 byte):
  0000 0000 0100 0000...0000 0000 1111 1111

  BCD data:
  0000 0000 0110 0100...0000 0010 0101 0101

  Gray code:
  0000 0000 0110 0000...0000 0000 1000 0000

## 35.7 Additional examples for Control FPWIN Pro7

Please find further example projects under `\Program Files\Panasonic Industry Control\Control FPWIN Pro7\Samples` (default installation folder).

If the examples are not installed, use the Install Shield of Control FPWIN Pro7 to configure your software installation.

1. Run setup from your Control FPWIN Pro7
2. Select “Modify” in the InstallShield wizard
3. Select examples to be installed in desired language

Use “Project” > “Open” or “Project” > “New” > “From file” to open the desired example or to import the example in an existing project.

### 35.7.1 Calculates the product of two-dimensional arrays

Creating a universal function that calculates the products of two-dimensional REAL arrays of varying sizes. It will also be double checked that all arrays lie in a permissible memory area (DT or FL) and that their dimensions meet the requirements of the multiplication of matrices.

### 35.7.2 Calculate the sum of all array elements

Creating a function block that calculates the sums of REAL arrays of varying sizes.

### 35.7.3 Calculates the trace of a three-dimensional array

Creates a universal function to calculate the spur of a matrix (sum of diagonal elements) of a three-dimensional array of varying sizes of the data type DINT. Thereby it will also be double checked that the array lies in a permissible memory area (DT or FL) and that its dimensions are the same.

### 35.7.4 Simulate an I/O panel (STRING)

A simple state machine with simulated IOP (I/O panel).

The IOP has three keys: **SwitchToNext** (R0), **CountUp** (R1), **CountDown** (R2) and displays the 16 ASCII-numbers of **Lcd_Buffer** (DT0-DT7).

### 35.7.5 State machine (LD)

Illustrates by means of small action-chain controls the possibilities to edit Boolean arrays or DUTs with Word operations.

### 35.7.6 Read, write IC card (REAL, DUT, FP10SH, FP2SH)

Creates a universal function or function block that helps you to write or read desired data objects from the IC card by using the basic instructions **F12_ICRD** or **F13_ICWT**.

Related topics

Memory device instructions

### 35.7.7 Read, write shared memory (REAL, DUT, FP10SH, FP2SH, FP2)

Creates a universal function or function block that helps you to write or read desired data objects from shared memory by using the basic instructions **F150_READ** or **F151_WRT**.

Related topics

Input, output and unit access instructions

## 35.8 Error codes

Tip

For PDF files, please refer to Panasonic Download Center

### 35.8.1 Table of syntax check error

In FPWIN Pro, syntax errors are detected by the compiler and are therefore not critical.

| Error code | Name | Operation status | Description and steps to take |
|---|---|---|---|
| E1 | Syntax error | Stops | A program with a syntax error has been written.<br>Change to PROG. mode and correct the error. |
| E2(* Note) | Duplicated output error | Stops | Two or more OT(Out) instructions and KP(Keep) instructions are programmed using the same relay.<br>Change to PROG. mode and correct the program so that one relay is not used for two or more OT instructions and KP instructions. Or, set the duplicated output to “enable (K1)” in system register 20. |
| E3 | Not paired error | Stops | For instructions which must be used in a pair such as jump (JP and LBL), one instruction is either missing or in an incorrect position.<br>Change to PROG. mode and enter the two instructions which must be used in a pair in the correct positions. |
| E4 | Parameter mismatch error | Stops | An instruction has been written which does not agree with system register settings. For example, the number setting in a program does not agree with the timer/ counter range setting.<br>Change to PROG. mode, check the system register settings, and change so that the settings and the instruction agree. |
| E5(* Note) | Program area error | Stops | An instruction which must be written to a specific area (main program area or subprogram area) has been written to a different area (for example, a subroutine SUB to RET is placed before an ED instruction).<br>Change to PROG. mode and enter the instruction into the correct area. |

| Error code | Name | Operation status | Description and steps to take |
|---|---|---|---|
| E6 | Compile memory full error(Available PLC: FPΣ/ FP-X/FP2SH/ FP10SH) | Stops | The program stored in the PLC is too large to compile in the program memory.<br>Change to PROG. mode and reduce the total number of steps for the program. |
| E7 | High-level instruction type error<br>(Available PLC: FPΣ/FP-X/FP2/ FP2SH/FP3/ FP10SH) | Stops | In the program, high-level instructions, which execute in every scan and at the rising edge of the trigger, are programmed to be triggered by one contact [e.g., F0 (MV) and P0 (PMV) are programmed using the same trigger continuously].<br>Correct the program so that the high-level instructions executed in every scan and only at the rising edge are triggered separately. |
| E8 | High-level instruction operand error | Stops | There is an incorrect operand in an instruction which requires a specific combination operands (for example, the operands must all be of a certain type).<br>Enter the correct combination of operands. |
| E9 | No program error(Available PLC: FP2SH/ FP10SH) | Stops | Program may be damaged.<br>Try to send the program again. |
| E10 | Rewrite during RUN syntax error | Continues | When inputting with the programming tool software, a deletion, addition or change of order of an instruction (ED, LBL, SUB, RET, INT, IRET, SSTP, and STPE) that cannot perform a rewrite during RUN is being attempted. Nothing is written to the CPU. |

Note

This error is also detected if you attempt to execute a rewrite containing a syntax error during RUN. In this case, nothing will be written to the CPU and operation will continue.

## 35.8.2 Table of self-diagnostic errors

Not all errors apply to all PLCs.

E20 - E39

| Error code | Name | Operation status | Description and steps to take |
|---|---|---|---|
| E20 | CPU error | Stops | Probably a hardware abnormality.<br>Please contact your dealer. |
| E21-E25 | RAM error | Stops | Probably an abnormality in the internal RAM.<br>Please contact your dealer. |

| Error code | Name | Operation status | Description and steps to take |
|---|---|---|---|
| E26 | User's ROM error | Stops | FP2, FP2SH, FP3, FP10SH:<br>ROM is not installed. There may be a problem with the installed ROM.<br>• ROM contents are damaged<br>• Program size stored on the ROM is larger than the capacity of the ROM<br>Check the contents of the ROM |
| | | | FP-X:<br>If the master memory cassette is mounted, the master memor cassette may be damaged. Remove the master memory, and check whether the ERROR turns off.<br>If the ERROR turned off, rewrite the master memory as its contents are damaged, and use it again.<br>If the ERROR does not turn off, please contact your dealer. |
| | | | FP0, FP-e, FP∑, FP1 C14, C16:<br>Probably an abnormality in the built-in ROM.<br>Please contact your dealer. |
| | | | All FP-Ms and FP1 C24, C40, C56, and C72:<br>Probably an abnormality in the memory unit or master memory unit.<br>Program the memory unit or master memory unit again and try to operate. If the same error is detected, try to operate with another memory unit or master memory unit. |
| E27 | Intelligent unit installation error | Stops | Intelligent units installed exceed the limitations (i.e. 4 or more link units).<br>Turn off the power and re-configure intelligent units referring to the hardware manual. |
| E28 | System register error | Stops | Probably an abnormality in the system register.<br>Check the system register setting or initialize the system registers. |
| E29 | Configuration parameter error | Stops | A parameter error was detected in the MEWNET-W2 configuration area. Set a correct parameter. |
| E30 | Interrupt error 0 | Stops | Probably a hardware abnormality.<br>Please contact your dealer. |
| E31 | Interrupt error 1 | Stops | An interrupt occurred without an interrupt request.A hardware problem or error due to noise is possible.<br>Turn off the power and check the noise conditions. |
| E32 | Interrupt error 2 | Stops | An interrupt occurred without an interrupt request.A hardware problem or error due to noise is possible.<br>Turn off the power and check the noise conditions. |

| Error code | Name | Operation status | Description and steps to take |
|---|---|---|---|
| | | | There is no interrupt program for an interrupt which occurred.<br>Check the number of the interrupt program and change it to agree with the interrupt request. |
| E33 | Multi-CPU data unmatch error | CPU2 stops | This error occurs when a FP3/FP10SH is used as CPU2 for a multi-CPU system.<br>Please contact your dealer. |
| E34 | I/O status error | Stops | An abnormal unit is installed.<br>Check the contents of special data register DT9036/ DT90036 and locate the abnormal unit. Then turn off the power and replace the unit with a new one. |
| E35 | MEWNET-F (remote I/O) slave illegal unit error | Stops | A unit, which cannot be installed on the slave station of the MEWNET-F link system, is installed on the slave station.<br>Remove the illegal unit from the slave station. |
| E36 | MEWNET-F limitation error | Stops | The number of slots or I/O points used for MEWNET-F exceeds the limitation.<br>Re-configure the system so that the number of slots and I/O points is within the specified range. |
| E37 | MEWNET-F I/O mapping error | Stops | I/O overlap or I/O setting that is over the range is detected in the allocated I/O and MEWNET-F I/O map.<br>Re-configure the I/O map correctly. |
| E38 | MEWNET-F slave I/O mapping error | Stops | I/O mapping for remote I/O terminal boards, remote I/O terminal units and I/O link unit is not correct.<br>Re-configure the I/O map for slave stations according to the I/O points of the slave stations. |
| E39 | IC memory card read error | Stops | When reading in the program from the IC memory card (due to automatic reading because of the dip switch 3 setting or program switching due to F14 (PGRD) instruction ):<br>• IC memory card is not installed.<br>• There is no program file or it is damaged.<br>• Writing is disabled.<br>• There is an abnormality in the AUTOEXEC.SPG file.<br>• Program size stored on the card is larger than the capacity of the CPU.<br>Install an IC memory card that has the program properly recorded and execute the read once again. |

E40 and above

| Error code | Name | Operation status | Description and steps to take |
|---|---|---|---|
| E40 | I/O error | Selectable | With FP3/FP10SH, communication error in the MEWNET-TR system has occurred.<br>For all other PLCs an abnormality in an I/O unit has been detected.<br>Check the contents of special data registers DT9002 and DT9003/DT90002 and DT90003 and the erroneous MEWNET-TR master unit or abnormal I/O unit (also expansion unit or application cassette). Then check the unit.<br>Selection of operation status using system register 21:<br>• to continue operation, set K1 (CONT)<br>• to stop operation, set K0 (STOP) |
| E41 | Intelligent unit error | Selectable | An abnormality in an intelligent unit.<br>Check the contents of special data registers DT9006 and DT9007/DT90006 and DT90007 and locate the abnormal intelligent unit. Then check the unit referring to its manual.<br>Selection of operation status using system register 22:<br>• to continue operation, set K1 (CONT)<br>• to stop operation, set K0 (STOP) |
| E42 | I/O unit verify error | Selectable | I/O unit wiring condition has changed compared to that at time of power-up.<br>Check the contents of special data registers DT9010 and DT9011/DT90010 and DT90011 and locate the erroneous unit.<br>Then check the unit and correct the wiring.<br>Selection of operation status using system register 23:<br>• to continue operation, set K1 (CONT)<br>• to stop operation, set K0 (STOP) |
| E43 | System watching dog timer error | Selectable | Scan time required for program execution exceeds the setting of the system watchdog timer.<br>Check the program and modify it so that FP2SH/FP10SH can execute a scan within the specified time.<br>Selection of operation status using system register 24:<br>• to continue operation, set K1 (CONT)<br>• to stop operation, set K0 (STOP) |
| E44 | Slave station connecting time error for MEWNET-F system | Selectable | The time required for slave station connection exceeds the setting of the system register 35.<br>Selection of operation status using system register 25:<br>• to continue operation, set K1 (CONT)<br>• to stop operation, set K0 (STOP) |

| Error code | Name | Operation status | Description and steps to take |
|---|---|---|---|
| E45 | Operation error | Selectable | Operation became impossible when a high-level instruction was executed.<br>Check the contents of special data registers DT9017 and DT9018/DT90017 and DT90018 to find the program address where the operation error occurred. Then correct the program.<br>Refer to the explanation of operation error and the instruction.<br>Selection of operation status using system register 26:<br>• to continue operation, set K1 (CONT)<br>• to stop operation, set K0 (STOP) |
| E46 | Remote I/O communication error | Selectable | **MEWNET-F communication error:**<br>A communication abnormally was caused by a transmission cable or during the power-down of a slave station.<br>Check the contents of special data registers DT9131 to DT9137/DT90131 to DT90137 and locate the abnormal slave station and recover the slave condition.<br>Selection of operation status using system register 27:<br>• to continue operation, set K1 (CONT)<br>• to stop operation, set K0 (STOP)<br>**S-Link communication error (with FP0-SL1 unit only):**<br>When one of the S-LINK errors (ERR1, 3 or 4) has been deteced,error code E46 (remote I/O (S-LINK) communication error) is stored.<br>Selection of operation status using system register 27:<br>• to continue operation, set K1 (CONT)<br>• to stop operation, set K0 (STOP) |
| E47 | MEWNET-F attribute error | Selectable | MEWNET-F communication error<br>A communication abnormally was caused by a transmission cable or during the power-down of a slave station.<br>Check the contents of special data registers DT9131 to DT9137/DT90131 to DT90137 and locate the abnormal slave station and recover the communication condition.<br>Selection of operation status using system register27:<br>• to continue operation,set K1<br>• to stop operation, set K0 |
| E50 | Backup battery error | Continues | The voltage of the backup battery lowered or the backup battery of CPU is not installed.<br>Check the installation of the backup battery and then replace battery if necessary.<br>By setting the system register 4 in K0 (NO), you can disregard this error. However, the BATT. LED turns on. |

| Error code | Name | Operation status | Description and steps to take |
|---|---|---|---|
| E51 | MEWNET-F terminal station error | Continues | Terminal station settings were not properly performed.Check stations at both ends of the communication path, and set them in the terminal station using the dip switches. |
| E52 | MEWNET-F I/O update synchronous error | Continues | Set the INITIALIZE/TEST selector to the INITIALIZE position while keeping the mode selector in the RUN position. If the same error occurs after this, please contact your dealer. |
| E53 | Multi-CPU registration error | Continues | Abnormality was detected when the multi-CPU system was used. Please contact your dealer. |
| E54 | IC memory card backup battery error | Continues | The voltage of the backup battery for the IC memory card is getting low. The BATT. LED does not turn on.<br>Charge or replace the backup battery of IC memory card. (The contents of the IC memory card cannot be guaranteed.) |
| E55 | IC memory card backup battery error | Continues | The voltage of the backup battery for IC memory card is getting low. The BATT. LED does not turn on.<br>Charge or replace the backup battery of IC memory card. (The contents of the IC memory card cannot be guaranteed.) |
| E56 | Incompatible IC memory card error | Continues | The IC memory card installed is not compatible with FP2SH/FP10SH. Replace the IC memory card compatible with FP2SH/FP10SH. |
| E57 | No unit for the configuration | Continues | MEWNET-W2<br>The MEWNET-W2 link unit is not installed in the slot specified using the configuration data.<br>Either install a unit in the specified slot or change the parameter. |
| E100 to E199 | Self- diagnostic error set by F148 (ERR)/ P148 (PERR) instruction | Stops | The self-diagnostic error specified by the F148 (ERR)/ P148 (PERR) instruction is occurred.<br>Take steps to clear the error condition according to the specification you chose. |
| E200 to E299 | | Continues | |

### 35.8.3 Table of communication check error

| Error code | Name | Operation status | Description and steps to take |
|---|---|---|---|
| E63 | PLC error mode(Available PLC: FP2/FP2SH/ FP3/FP10SH) | Stops | Transfer was attempted in the RUN mode.<br>Switch the mode and execute once again. |
| E64 | No ROM/RAM error(Available PLC: FP2/FP2SH/ FP3/FP10SH) | Stops | An abnormality occurred when loading RAM to ROM/IC memory card. There may be a problem with the ROM or IC memory card.<br>• When loading, the specified contents exceeded the capacity (256 KB).<br>• Write error occurs.<br>• ROM or IC memory card is not installed.<br>• ROM or IC memory card does not conform to specifications.<br>Check the contents of the ROM or IC memory card. |
| E65 | Protection error | Stops | Transfer was attempted during ROM operation or IC memory card operation.<br>Switch the mode and execute once again. |
| E66 | PLC write error address error(Available PLC: FP2/FP2SH/ FP3/FP10SH) | Continues | In the programming tool software, program editing is being attempted by online access, but the program is not in agreement. (The program disagreement lies in another block.)Check the program. |
| E68 | Rewrite during RUN error(Available PLC: FP2/FP2SH/ FP3/FP10SH) | Continues | When inputting with the programming tool software, editing of an instruction (ED, SUB, RET, INT, IRET, SSTP, and STPE) that cannot perform a rewrite during RUN is being attempted. Nothing is written to the CPU. |

### 35.8.4 Table of Ethernet communication error codes

List of error codes that may occur during communication with FTP, HTTP, or SMTP clients.

#### FTP error codes

| Error code | Description |
|---|---|
| 226 | Normal end |
| 421 | Service not available, closing control connection |
| 425 | Cannot open data connection |
| 426 | Connection closed; transfer aborted |

| Error code | Description |
|---|---|
| 450 | Requested file action not taken; file unavailable |
| 451 | Requested action aborted: local error in processing |
| 452 | Requested action not taken: insufficient storage space in system |
| 500 | Syntax error |
| 501 | Syntax error in parameters or arguments |
| 502 | Command not implemented |
| 503 | Bad sequence of commands |
| 504 | Command not implemented for that parameter |
| 530 | Not logged in |
| 532 | Need account for storing files |
| 550 | Requested file action not taken; file unavailable |
| 551 | Requested action aborted: page type unknown |
| 552 | Requested file action aborted: Exceeded storage allocation |
| 553 | Requested action not taken: File name not allowed |
| 1XXX | An error occurred during file deletion after transfer (not to be retried) |
| 9XX | Client service error |

### HTTP error codes

| Error code | Description |
|---|---|
| 2XX | Normal end |
| 300 | Multiple choices<br>For example, when multiple pages can be used. |
| 301 | Moved permanently<br>This address was moved to another address. Future requests should be directed at the new URI (Uniform Resource Identifier) |
| 302 | Moved temporarily / Found |
| 303 | See other<br>Refer to another page. |
| 304 | Not modified<br>Although the access was permitted, the target document has not been updated. |
| 305 | Use proxy<br>Only the access via the proxy can be permitted (proxy address is provided in the response). |
| 307 | Temporary redirect<br>This address temporarily belongs to another address. Future requests should be directed at the old URI. |
| 400 | Bad request<br>Request cannot be processed due to an error such as a typing mistake. |

| Error code | Description |
|---|---|
| 401 | Unauthorized<br>The authentication has failed, e.g. a wrong password has been entered. |
| 403 | Forbidden<br>The user does not have access rights. |
| 404 | Not found<br>The page does not exist or the server is down. |
| 405 | Method not allowed<br>A request of an unpermitted method type was received, e.g. a GET request was sent for data that require a POST request. |
| 406 | Not acceptable<br>As a result drawn from the Accept header, unacceptable content was included. |
| 407 | Proxy authentication required |
| 408 | Request timeout |
| 409 | Conflict<br>The request could not be processed because of conflict in the current state of the resource, e.g. there is an editing conflict due to multiple simultaneous updates. |
| 410 | Gone<br>The resource requested is no longer available and will not be available again in the future. |
| 411 | Length required<br>The request was rejected because it did not contain a specified length for its content. |
| 412 | Precondition failed<br>The server fails to meet one of the preconditions that are listed in the request header fields. |
| 413 | Payload too large<br>The request was rejected because its size is larger than the server can process. |
| 414 | URI too long |
| 415 | Unsupported media type<br>The requested service was rejected by the server because the requested resource is in an unsupported format for the requested method. |
| 416 | Range not satisfiable<br>The client requests a portion of the file, but the server is unable to supply that portion. |
| 417 | Expectation failed<br>The sever fails to meet the requirements of the Expect request-header field. |
| 500 | Internal server error.<br>An error occurs in CGI script, etc. |
| 501 | Not implemented<br>The function required for executing the request is not supported. |
| 502 | Bad gateway<br>An incorrect response was received when the server acting as a gateway or proxy attempted to execute a request. |

| Error code | Description |
|---|---|
| 503 | Service unavailable<br>Request cannot be handled because the server is overloaded, down for maintenance or otherwise unavailable. |
| 504 | Gateway timeout |
| 505 | HTTP version not supported |
| 9XX | Client service error |

### SMTP error codes

| Error code | Description |
|---|---|
| 0 | Normal end |
| 421 | Domain service not available, closing transmission channel |
| 450 | Requested mail action not taken: mailbox unavailable (temporarily) |
| 451 | Requested action aborted: local error in processing |
| 452 | Requested action not taken: insufficient storage space in system |
| 500 | Syntax error, command unrecognized<br>Caused by e.g. a typing mistake or because the command line is too long. |
| 501 | Syntax error in parameters or arguments |
| 502 | Command not implemented |
| 503 | Bad sequence of commands |
| 504 | Command parameter is not implemented. |
| 550 | Requested mail action not taken: mailbox unavailable (permanently). |
| 551 | User not local, please try <forward path><br>Transfer is interrupted because the sender and the recipient are not locally hosted by the server. |
| 552 | Requested mail actions aborted: Exceeded storage allocation<br>The recipient's mailbox is overflowing and does not accept any more mails. |
| 553 | Requested action not taken: mailbox name not allowed<br>One or more of the addresses in the TO, CC or BCC field is invalid, e.g. due to a typing mistake. |
| 554 | Transaction has failed<br>This is a permanent error, the message will not be sent again. |
| 9XX | Client service error |

### 35.8.5 Table of Modbus/MEWTOCOL communication error codes (FP7 only)

List of error codes that may occur during Modbus/MEWTOCOL communication.

| Error code | Description |
|---|---|
| 0 | Normal completion |
| 1 | Communication port is used for master communication |
| 2 | Communication port is used for slave communication |
| 3 | No. of master communication instructions that can be used at the same time is exceeded |
| 4 | Transmission timeout |
| 5 | Timeout while waiting for a response |
| 6 | Error in received data<br>This error occurs when an abnormal telegram is received, e.g. when there is a format error in the header. In this case, the received data is discarded. |
| 7 | Insufficient area reserved in I/O map<br>This error occurs when the number of Ethernet connections exceeds 16 (Advanced Ethernet communication). Check whether the word area reserved for communication control flags is big enough for the number of Ethernet connections. |
| 8 | Send buffer currently in use<br>This error may occur for FP7 PLCs with firmware version 4.57 or later. |
| 9 | Master unit station number not set |
| 21 | NACK error |
| 22 | WACK error |
| 23 | Duplicated unit number |
| 24 | Transmission format error |
| 25 | FP7 multi-wire link unit hardware error |
| 26 | Unit number setting errror |
| 27 | Packet not supported |
| 28 | No response |
| 29 | FP7 multi-wire link unit hardware error |
| 30 | Transmission timeout |
| 31–39 | FP7 multi-wire link unit hardware error |
| 41 | Format error |
| 60 | Parameter error |
| 61 | Data error |
| 91 | Missing expansion unit error |
| 8001 | Function code error |

| Error code | Description |
|---|---|
| 8002 | Starting address outside the permissible range |
| 8003 | Number of data outside the permissible range |

# 36 Record of changes

ACGM0313V5EN, 2025.02

Complete update in accordance with software version 7.7.3.0. For the latest release notes of Control FPWIN Pro7, visit https://infohub.industry.panasonic.eu/documentation/fpwin/en/#/topic/t-0000014305.

ACGM0313V4EN, 2021.06

Complete update in accordance with software version 7.5.2.0. For details on the new information, see the section new in this version 7.5.2.0 in the online help.

ACGM0313V3EN, 2012.12

Complete update in accordance with software version 6.4.1.0 For details on the new information, see the section new in this version 6.4.1.0 in the online help.

ACGM0313V2EN, 2012.07

Complete update in accordance with software version 6.4. For details on the new information, see the section new in this version 6.4 in the online help.

ACGM0313V3EN, 2011.03

First edition

# Index


## A

ABCD 1941
ABIN 1941
AHEX 1941
ATAN2 1941

## B

BAND 1941
BCC 1941
BCD 1941
BCDA 1941
BCDADD 1941
BCDDEC 1941
BCDDIV 1941, 1941, 1941
BCDINC 1941
BCDMUL 1941
BCDSUB 1941
BCMP 1941
BCU 1941
BDIS 1941
BIN 1941
BINA 1941
BITL 1941
BITR 1941
BKMV 1941
BTI 1941
BTM 1941
BTT 1941
BUNI 1941

## C

CCOPY 1941
CDTRD 1941
CDTWT 1941
CFDEL 1941
CFLS 1941
CFREEK 1941
CMKDIR 1941
CMP 1941
CMPR 1941
CMPW 1941
CMV 1941
COLM 1941
COMB 1941
Communication modes 528
COPY 1941
COSH 1941
CPR 1941
CRC 1941
CRD 1941
CRD1 1941
CREN 1941
CRMDIR 1941
CT_FB 971
CWT 1941

## D

DECO 1941
DEFBUF 1941
DEG 1941
DGT 1941
DI 1941
DIST 1941
DIVMOD 1941
DTR 1941

## E

EI 1941
ELEFT 1941
ELEN 1941
EMIDR 1941
ENCO 1941
ERIGHT 1941
ERR 1941
ESADD 1941
ESCMP 1941
ESREP 1941
ESSRC 1941
EVENTC 1941
EVENTT 1941
EZPID 1941

## F

FABS 1941
FIFR 1941
FIFW 1941, 1941
FILTR 1941
FINT 1941
FIX 1941
FLT 1941
FRINT 1941

## G

GBIN 1941
GETSTNO 1941
GPRECV 1941
GPSEND 1941
GRY 1941

## H

HEXA 1941
HSC, High Speed Counter Instructions 1156, 1156

## I

IMASKICLR 1941
INC 1941
INT 1941
INV 1941

## L

LIFR 1941
LIMT 1941
LINE 1941
LOGED 1941
LOGST 1941

## M

MEAN 1941
MV 1941
MVN 1941

## P

P13_EPWT 1409
PERRD 1941
PID 1941
PMGET 1941, 1941
PMSET 1941
POSSET 1941
PSTRD 1941
PulseOutput_Channel_Configuration_DUT 1634
PWR 1941

## R

RAD 1941
RAMP 1941
RCL 1941
RCR 1941
RDET 1941
RECV 1941
RTU Master/Slave 370

## S

SCAL 1941
SECTM 1941
SEGT 1941
SEND 1941
SFC Control Instructions 1737
SINH 1941
SMPL 1941
SORT 1941
SPTM 1941
SQR 1941
SRC 1941
SWAP 1941
System Variables for special relays or special data registers 952

## T

TANH 1941
TMSEC 1941

## U

UCLR 1941
UNIT 1941

## W

WDTRES 1941
WIN 1941
WSHL 1941
WSHR 1941

## X

XCH 1941

## Z

ZONE 1941