Motion Control PMC

pilz

more than automation

safe automation

PMCprimo SoftPLC

User Manual – Item No. 21 471-06

1 General conditions

1.1 Copyright

1.2 Notes

Pilz GmbH & Co. KG reserves the right to make amendments to this document at any time.

The examples given serve only as illustrations. No guarantee is given for their suitability in particular applications. Although the utmost care has been taken in the production of this document, no liability can be accepted for any mistakes that it may contain. We welcome any suggestions for the improvement of our products, or documentation.

We reserve the right to make technical changes, which lead to the improvement of the product.

1.3 Previous editions

Edition	Comments
V1-18-02-2003	First edition: Valid from PMCprimo software version >=1.008
V2-15-05-2003	Valid from PMCprimo software version >=2.000
V3-30-04-2004	Valid from PMCprimo software version >=2.004
V4-10-02-2005	Valid from PMCprimo software version >=2.005
V5-07-06-2005	Valid from PMCprimo software version >=2.006
V6-05-12-2005	Revision

2 Contents

3 Contents of illustrations

4 Abbreviations and symbols

Abbreviation	Meaning/Description
PMCprimo	Registered trade-mark Pilz GmbH & Co. KG
PMCtendo	Registered trade-mark Pilz GmbH & Co. KG
Hiperface®	Registered trade-mark Max Stegmann GmbH

Symbols	Meaning/Description
	This symbol indicates the possibility of imminent danger to life and limb. Failure to observe these warnings can lead to serious or life-threatening injury.
	This symbol indicates important instructions for the correct operation of the drive. Failure to observe these instructions can lead to failures of the drive or the connected system.
	This symbol indicates application tips and information of an especially useful nature. These will help achieve the optimal use of all the available functions.

5 Introduction

This manual describes the specialties and the functional library of the SoftPLC according to the IEC-61131 standard. It can be used as a supplement to the separate CoDeSys user manual which describes the general programming and operation.

Please carefully read this user manual. Reference manual for all commands is the german version.

PMCprimo motion control systems are available in different types and can control more than 700 axes in a network. The coordination of the separate axes in PMCprimo is realized over the host level. Every PMCprimo-device can be used as a host system.

PMCprimo is programmed in a high level language or IEC 61131-3 (CoDeSys). Central communication medium is a terminal software (see user manual “Motion Control Tools”) on your PC, that will be connected via a RS232 or Ethernet interface. All parameterisations and user programs can be saved in flash memory or a Compact-Flash-Card.

Software functions for movement of servo axes

With the described PMCprimo-commands the parameters will be set and functions for the respective task will be programmed.

The wide range of commands allows the realization of complex applications. PMCprimo has axis spanned commands, this means that a separate command can affect several axes.

Reference commands allow the automatic initialisation and a reference error correction at a running machine. The reference error can be compensated variable, for example with defined ramp or time functions.

By this, divergences in product or machine parameters can be corrected flexible (cycle based).

Free definable master-slave-relationships allow the user to define any speed profiles for motors. The usage of jerk less movement flow (for example modified sine) reduces the mechanical load at the machine as much as possible.

PMCprimo can fit motors with electronic gear functions just by software. The relation between motor positions can be defined freely by the user, additionally to the electronic gear function (linear and non linear correlation with a tabular position assignment). There are multiple of VDI defined movement kinds for your choice. Tabular position assignments can replace mechanical cams and mechanical gearboxes.

When needed PMCprimo can generate new maps with the internal motion generator during machine operation. Product dependent machine changeovers can be done by the push of a button.

Beside several possibilities to synchronize machine axes, there are also many commands for absolute, relative or endless positioning of power transmissions available.

All axes can operate in virtual mode (electronic master axis), for example for initial operation or as help axes.

Hardware functions

PMCprimo is intended for use with digital incremental position encoders which provide two signals in
quadrature. This allows the system to measure both the distance and direction of motion of the motor, thus providing the closed-loop feedback information for the channel. The encoder input interface circuit multiplies the resolution of the encoder by four, such that each complete cycle of the encoder signals represents four counts. PMCprimo includes full isolation of the encoder input signals, and are designed for use with encoders having differential line driver outputs. This is get best performance and noise rejection in an industrial environment. You can also use SSI-, Hiperface-, or CANopen-encoder instead of incremental position encoders.

PMCprimo has digital input and digital output lines (the number depends to the type of PMCprimo), which may be used in various ways. Additional has PMCprimo analogue inputs and outputs. All digital input lines gives with a change of their state an interrupt. The user defines what happens with the interrupt. The state of the input line or the value of the analogue input could be checked in sequences. The state of the outputs is also controlled by sequences. Outputs may be explicitly set and cleared, and can be used to control external relays or valves, or just for status indication.

The inputs and outputs can be used as an interface to a PLC The digital input and output lines are fully isolated and are compatible with 24 V logic signals. For comfortable communication with a PLC or any other host computer a RS232 interface, MODBUS, several field bus systems or Ethernet are available.

The analogue outputs give signals, ranging from -10 V to + 10 V.

Applications:

Typical applications for PMCprimo motion control systems are:

- Packaging machinery
- Modular machines
- Printing and paper machineries

Handling systems

6 Manufacturer's declaration / Safety Instructions

6.1 Manufacturer's declaration

Controls are not machines within the scope of the Machinery Directive 98/37/EG, but components for installation into machines.
An initial start-up is prohibited until it has been noticed that the electrical equipment or machinery in which the controls are incorporated correspond to the requirements of the EG-regulations.

6.2 Safety Instructions

Warning!
Do not touch live mains and components – potentially fatal.
During initial operation it has to be assured that there will be no danger to personnel and damage to machinery or equipment.
For this reason the following safety precautions must be taken.
Only qualified and well-trained specialists should work on the units to avoid any injury to personnel or damage to machinery.

Warning!
As a general principle, electronic appliances are not failure-free.
The Installation and Operating Instructions must be read carefully and all safety regulations observed before installation and initial operation as incorrect handling can cause injury to personnel and damage to machinery.

Please note:

- Only qualified and well-trained specialists who are familiar with the transportation, installation, initial operation, maintenance and operation of the units, as well as with the relevant standards, may carry out the corresponding works.
- Technical data and information (Type tag and documentation) must be observed.
- **ESD-Danger!**
  Control units contain components which will be damaged by electrostatic discharge if handled incorrectly.
  Therefore: - before touching the control unit:
  „Ensure that your own body is earthed"
  Always place the control unit on a conductive surface with good earth connection.
  Avoid touching contacts, components and plug connections.
- Never disconnect electrical connections when the drive is connected to mains power.
- Do not open the covers and switch enclosures when the drive is running.
- Before opening the units switch off mains power.
  Wait at least 5 minutes before opening.
  As a precaution test the intermediate circuit.
  Open only when it is < 40 V.

7 General description

The option PMCprimo-SoftPLC is a supplement to the existing software functionality by a IEC-61131 programmable software PLC (PLC = programmable logic control).

Release this software functionality is made by the PMCprimo command SK:

```
Series number: 000102
Installed Software keys:
Motion: 49a84d
Ethernet: 03fce7
SoftPLC: 178195

New Key?
```

A releasing has been carried out by an input of the code: 178195. This code is different for every unit. The included CoDeSys licence label must be attached at the unit in case of a subsequent activation.

Note: The PMCprimo commands are described in a separate programming user manual.

Additionally the programming tool CoDeSys (Controller Development System) is necessary which is always required for a programming. This tool serves for a programming in the various IEC-61131 languages (IL, LD, ST, FBD, etc.) and directly generates an assemble code at the processor of the control unit. Without this program it is not possible to carry out modifications. When installing the CoDeSys program additionally a so-called target information data is installed which describes the characteristics of the PMCprimo-control.

Change since version **2.001**:
The RAM memory is now allocated dynamically from the PLC. This means the PLC program is not limited to 256 KB or 2MB. The PLC program can be as big as free memory is available.

8 News in PMCprimo firmware version 2.005

Function block	PMCprimo command	News/Change	Page
		New primo.lib: Change of limits of some function blocks:	30
SetAlignmentAcceleration	AA	diAcceleration: DINT, to udiAcceleration: UDINT	109
SetMapBaseAdvance	BA	uiPhase: UINT, to diPhase: DINT	119
SetMonitorOutputGain	KM	iValue: UINT, to diValue: DINT	39
SetClutchWindow	CI	diValue: DINT, to uiValue: UINT	115
SetReferenceErrorLimit	LR	udiValue: UDINT, to uiValue: UINT	262
SetClutchLength	CL	diValue: DINT, to udiValue:DINT	111
SetReferenceFalseHighLimit	FH	uiValue: UINT, to udiValue: UDINT	264
SetReferenceFalseLowLimit	FL	uiValue: UINT, to udiValue: UDINT	266
SetReferenceTrueHighLimit	ZH	uiValue: UINT, to udiValue: UDINT	266
SetReferenceTrueLowLimit	ZL	uiValue: UINT, to udiValue: UDINT	266
SetPositionOutputHysteresis	PH	uiValue: UINT, to udiValue: UDINT	170
SetReferenceCorrectionVelocity	RV	usiValue: USINT, to udiValue: UDINT	260
SetReferenceHoldOffTime	RH	usiValue: USINT, to uiValue: UINT	269
SetVelocityOutputHysteresis	VH	uiValue: UINT, to udiValue: UDINT	172
SetMapPositionTimeout	MT	New range of values: 0 to 1000	131
SetMapOptionsWord	MW	Alignment acceleration also with speed mapping	127
SetMapOptionsWord	MW	New Spezial case bit 0 and bit 2 are set simultaneously	127
SetModbusAddress	MU	New function block	310
		Definition of new tasks	
		Number of the global variables g_iModbusData increased from 1024 to 2048	21
DisplayPositionOverflowCounter	BC	Display of the position overflow counter	56
DisplayPositionBound	SB	Display of the actual position bound	57
SetMapOptionsWord	MW	Now it is possible to interrupt speed mapping with stop to position.	127
SetScaleMapping	SM	New range of values for uiCounter and uiDevider: 400.000	136
DefineTimerOutput	TC	Definition of a timer-/counter output	161
SetReferenceAdvanceFactor	RN	New range of values: ± 65.535	258
WaitForRelativePosition	WR	New range of value: ±2.000.000.000	305

9 Installation of the software

The included CD for the CoDeSys programming tool is self-installing. In case the Autoplay Option is switched-off on the computer, you must manually start the installation under CoDeSysV2.3/Setup.exe.
All required components as for example the target information data and the PMCprimo library are copied.

Herewith all required components are available.

9.1 Usage of directories

Different subdirectories will be added in the installation directory. Here a short description:

Directory:

Doku	PDF documents
Help	Files for online help in different languages
English	
French	
German	
Library	Additional libraries and function blocks
PLCConf	GSD and EDS - files for Profibus and CANopen
Projects	Directory for the projects
Target	Files for the different target descriptions for any manufacturer
Primo	The files necessary for PMCprimo

If additional GSD files should be added then they have to be copied in the directory Library/PLCConf.

The installation program automatically installs all for PMCprimo necessary files. This can be done also with the program InstallTarget manually.

10 First steps

Please start the program CoDeSys for programming the control. A new window opens. If the program is started the first time or a new program should be started, please click in the menu data file to new. Now a window is opened in which the target system must be set.

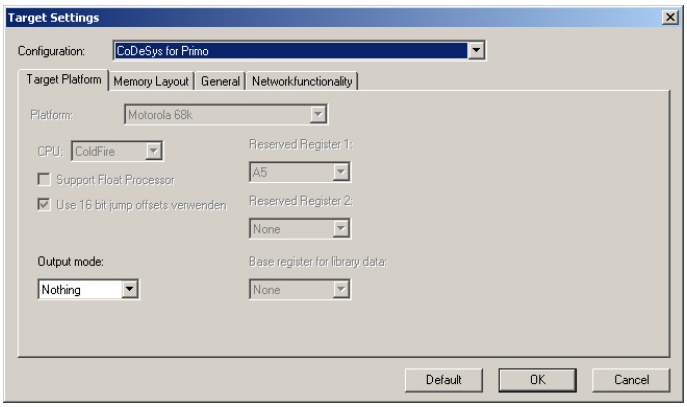


Illustration 1: target system

Here „CoDeSys for Primo“ must be set and then the OK switching section must be clicked. Normally no further settings required.

Afterwards this window appears, in which the first module „PLC_PRG“ (as a Program !) is defined.

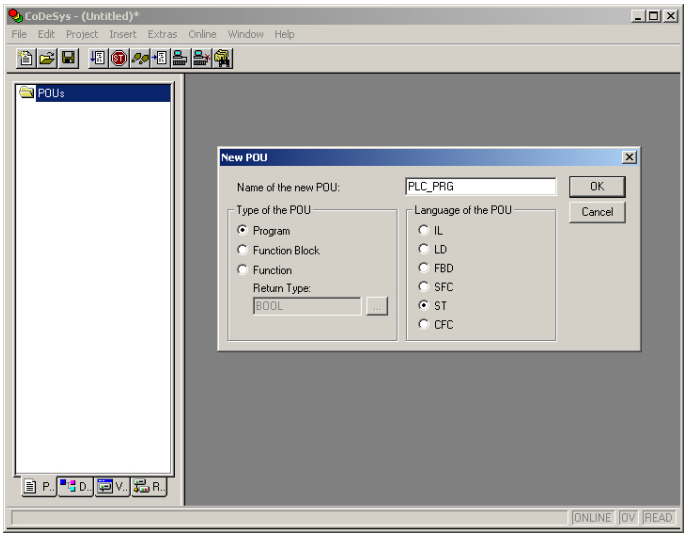


Illustration 2: module PLC_PRG

The next, you should define the control configuration. It is made in the resources. In addition to this click the right card tab below on the right and then click twice on the control configuration.

11 Control configuration

The connected Hardware is defined in the control configuration.

First of all automatically the Host is registered after preparation of a new project. For this 4 global variables HostInput1, HostInput2, HostOutput1 and HostOutput2 are defined.

The variables have different meanings dependent on the Hardware.

11.1 PMCprimo Drive:

Version < 2.004:

HostInput1 (Word) corresponds to 12 digital inputs, the remaining 4 Bits are not used.
HostInput2 (Word) presents 8 virtual inputs, the remaining 8 Bits are not used..
HostOutput1 (Word) is provided for 8 digital and 8 virtual outputs.
HostOutput2 (Word) is not used..

From version 2.004:

HostInput1 (Word) corresponds to 12 digital inputs, the remaining 4 Bits are not used.
HostInput2 (Word) presents 16 virtual inputs.
HostOutput1 (Word) is provided for 8 digital and the first 8 virtual outputs.
HostOutput2 (Word) is provided for 16 virtual outputs.

For compatibility the first 8 virtual outputs are accessible with HostOutput1 and HostOutput2. For new projects the HostOutput2 shall be used for virtual outputs.

11.2 PMCprimo 2+2:

HostInput1 (Word) corresponds to 16 digital inputs.
HostInput2 (Word) presents 8 virtual inputs, the remaining 8 Bits are not used.
HostOutput1 (Word) is provided for 16 digital outputs.
HostOutput2 (Word) corresponds to 8 virtual outputs again, the remaining 8 Bits are not used again.

11.3 PMCprimo 16+:

HostInput1 (Word) corresponds to 16 digital inputs.
HostInput2 (Word) presents 16 virtual inputs.
HostOutput1 (Word) is provided for 16 digital outputs.
HostOutput2 (Word) is provided for 16 virtual outputs

11.4 PMCprimo Drive2:

HostInput1 (Word) corresponds to 12 digital inputs, the remaining 4 Bits are not used.
HostInput2 (Word) presents 16 virtual inputs.
HostOutput1 (Word) is provided for 8 digital and the first 8 virtual outputs.
HostOutput2 (Word) is provided for 16 virtual outputs.

For compatibility the first 8 virtual outputs are accessible with HostOutput1 and HostOutput2. For new projects the HostOutput2 shall be used for virtual outputs. It is the same usage as for PMCprimo Drive and therefore projects are compatible with both hardware versions.

Therefore always 2 words for the inputs and 2 words for the outputs are used with all systems.

The name HostInput and HostOutput can be changed at any time (double click on the name).

If additionally there are nodes in the network, those must be advised to the PLC. For this, new sub elements must be added to the control configuration. First click „Module primo [Slot]“ in the list with the left mouse button and then press the right mouse button. A context menu appears, to which a sub element can be added.

The module PMCprimo indicates the configuration of the control, i.e. here it is set how many nodes are available in the network.

„primo Node Input/Output“ adds the information for a further node.

The global variables are called NodeInput and NodeOutput. The holding is identical as with the Host.

In case further nodes are added, you have to observe that the variables NodeInput und NodeOutput are not named, as otherwise the same variable names refer to different memory ranges and then an error message is indicated during compiling.

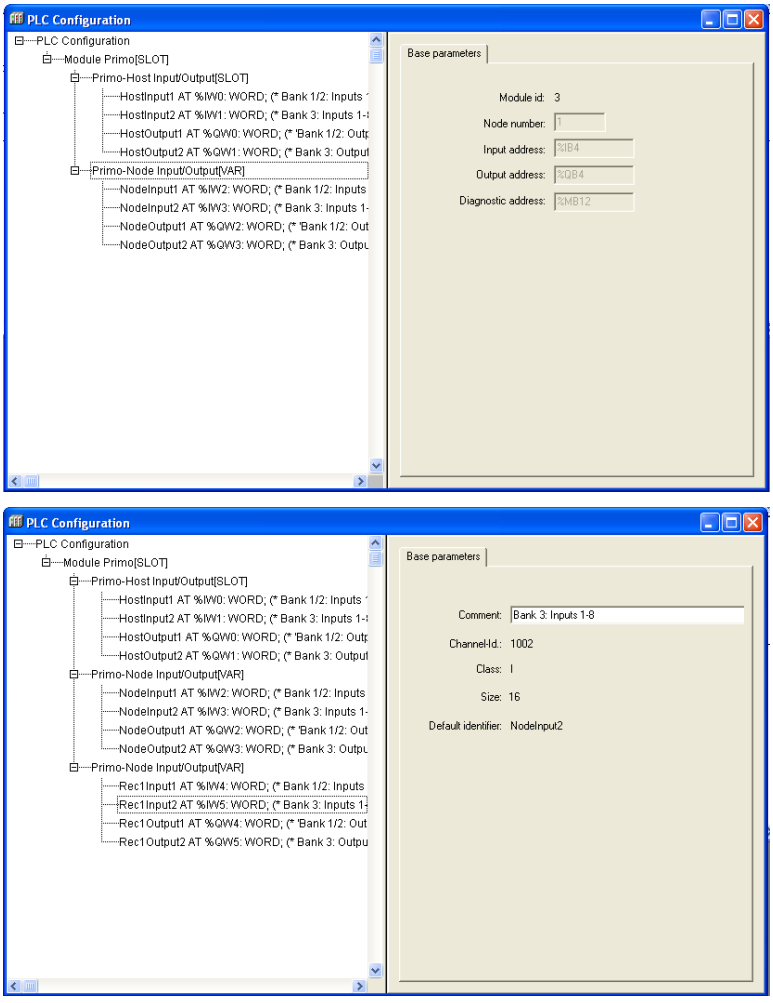


Illustration 3: Renamed inputs

When the right mouse button is pressed via PLC configuration, another context menu appears, in which further modules can be added.

Following modules are currently supported:

11.5 PMCprimo-CAN Input/Output:

Firmware-Version < 2.003

A sub-element is added for the communication with a CAN-I/O extension. Automatically global variables are arranged for 64 inputs (8 Bytes) and 64 outputs. This is also the maximum number of inputs and outputs. This storage range is also pre-defined, if there are not so many input or output modules are available. In this case the corresponding Bytes are not occupied.

Firmware-Version ≥ 2.003:

After the sub-elemente PMCprimo-CAN is added then it is possible to add with the context menu additional inputs and outputs to this element. With the menu PMCprimo CAN Input/Output 8 inputs and 8 outputs are added. The maximum number is 256 inputs and 256 outputs.

Only one bus coupler is supported with 256 inputs and outputs. If a second sub-element PMCprimo-CAN is added then it is ignored.

11.6 Profibus-DP enhanced Master:

A sub-element for the Profibus-DP Master card is included. This card can be installed into the PMCprimo 2+2 or 16+ as option and can be used for communication with further input/output modules, HMI-units or further Profibus units. If you click with the right mouse button over the list entry Profibus-DP enhanced Master a context menu with possible Profibus units appears. This information is taken from the GSD-files.

In order to install new GSD-files copy the files to C:\Codesysforautomationalliance\library\plcconf or into the installation list given by you. With the next start of the programming tools appears a further entry in the context menu.

For the Profibus-DP Master you must set the speed of the Profibus (Standard 9.6 kBits/s) under the Bus Parameters.

For the Profibus-DP Slave units the used inputs/outputs are to be set exactly in such a way as the Hardware is configurated. If you use for example 64 inputs, 8 Bytes must be set as input Byte. If the number does not coincide, the Profibus cannot be started and a communication cannot take place.

Diagnosis Bytes of the Profibus-DP Masters

The Profibus-DP Master writes diagnosis data into the diagnosis range. This address can also be determined in the control configuration for the Master. The first address (for example %MB8) is determined. 4 Bytes are written by the Profibus-Master which have the following meaning:

Byte 0: Globale Bits (Example %MB8)

Bit	Meaning
0	0: OK 1: Parameterisation failure
1	0: OK 1: unit went to Auto Clear Modus, as there is an error on the removed unit
2	0: OK 1: At least one node does not exchange any data or signals failure.

Byte 1: Master- und Network status (example %MB9)

Value	Meaning
0	Offline
64 = 16#40	Stop
128 = 16#80	Clear
192 = 16#C0	Operate

Byte 2: Failure of removed nodes (example %MB10)

Value	Meaning
0	No failure
1 - 254	Address of the smallest node, which indicates a failure. (detailed failure cause see Byte 3)
255	Internal failure in the Master (detailed failure cause see Byte 3)

Byte 3: Failure cause with external failures (Byte 2 from 1 up to 254)
The node address of the defective unit is indicated in Byte 2..

Value	Meaning	Removal
0	No failure	
3	Function in nodes is not available	Check, if unit is compatible to the Profibus-DP standard or if the correct GSD was used.
9	No response data	Check bus cable
17	No feedback of the slave node	Check bus cable and bus address of the slave node
18	The unit is not integrated in the Tokenring	Check the highest station address of the Profibus Master

Byte 3: Failure cause with internal failure (Byte 2 is 255)

Value	Meaning	Removal
0	No failure	
50-53	Internal failure	Contact company Pilz
54	No master configuration	Check the configuration
55	Defective parameter value for the master configuration	Contact company Pilz
56	No configuration for the removed node	Check the configuration
57	Defective parameter value for the removed nodes	Contact company Pilz
58	Double removed node address	Check the node addresses
202	No free segment	Contact company Pilz
212	Failure when reading the configuration data	Check the configuration
213	System error	Contact company Pilz

Diagnostic Bytes of the Profibus-DP Slave

A Profibus-DP Slave also writes diagnostic data in the diagnostic address range. The address can be determined but not set in the control configuration, as the required storage range is determined from the GSD-file. The first 6 Bytes are standard and have the following meaning.

Byte 0	Bit 0: Slave does not respond (is internally located by the DP-Master) Bit 1: Slave is in a high running position (parameter and configuration is evaluated Bit 2: Configuration failure Bit 3: Ext_Diag_Data are available (from Byte 6 on) Bit 4: Function is not supported Bit 5: False response from the Slave (is internally located by the DP-Master) Bit 6: Parametric failure Bit 7: Slave is in a data exchange with another Master (is internally located by the Profibus-DP Master)
Byte 1	Bit 0: Slave must be brought in a new parametric condition Bit 1: Slave has static diagnosis Bit 2: 1 Bit 3: DP-Watchdog is active Bit 4: Slave is in a Freeze-Mode Bit 5: Slave is in a Sync-Mode Bit 6: Reserved Bit 7: Slave is de-activated (is internally located by the DP-Master)
Byte 2	Bit 0-6: Reserved Bit 7: Too many Ext_Diag_Data
Byte 3	Station address from the Master, carried out with the data exchange
Byte 4,5	Ident-Number

Is the Bit 3 located in the Byte 0, there are further diagnosis data available. These are stored from the 6. Byte on and depend on the manufacturer. Therefore the meaning is to be determined in the documentation of the manufacturer.

11.7 Profibus-Slave

If the module Anybus-S PDP is added to the PLC configuration then the SoftPLC communicates direct with the Profibus board inside of the PMCprimo 2+2 or PMCprimo 16+. For the settings different additional inputs are required. The setting of the station address is ignored because this is done with the switches on the board. With the inputs and outputs the number of inputs and outputs have to be adjusted. This must exactly correspond to the configuration inside the Profibus master. Important notice: the inputs are from the view of the master. This means outputs in CoDeSys (%Q).

If the module is integrated in the PLC configuration then the Profibus board is used only by the SoftPLC. The normal connection to PMCprimo with bus variables $B is not used any longer. The settings in the configuration menu (CD command) are not used.

For the SoftPLC the Profibus board is an expansion of inputs and outputs.

11.8 Profibus-Slave IC

With the PMCprimo 16+ / Drive2 it is possible to mount a special Profibus-IC. With this module maximum 32 bytes input and 32 bytes output data is possible. As with the Profibus-Slave this board is used normally by PMCprimo with bus variables. For this in the configuration menu different settings are used (size of data, offset and station address). It is also possible to enable a direct communication. If the module Anybus-IC PDP is added to the PLC configuration then the SoftPLC uses this Profibus-IC as an input- and output expansion. The station address has to be set in the corresponding input mask because there is no hardware switch on board. The setting for the inputs and outputs works like all other Profibus modules.

12 Communication with the control

12.1 Kind of connections

Two possibilities are available for the communication with the control depending on the used Hardware.

PMCprimo Drive	Only serial communication
PMCprimo 2+2	Ethernet and serial communication
PMCprimo 16+	Ethernet and serial communication
PMCprimo Drive2	With optional expansion board Ethernet otherwise only serial communication

Ethernet:

An IP-address must be set in the control and the option Ethernet must be released via a Softwarekey.

The option Ethernet is always released with PMCprimo 16+ and PMCprimo Drive2. No Softwarekey is necessary.

This is done by means of a PMCprimo CD- and SK-Command.

```
0.1: SK <RETURN>
0.1: Series number: 000102
Installed Softwarekeys:
Motion: 49a84d
Ethernet: 03fce7
SoftPLC: 178195

New Key?

The control must be switched-OFF and ON after the option Ethernet has
been released.

0.1: cd <return>
0.1:
Actual configuration:
Operate Mode: STANDALONE
Actual IP address 192.168.0.6
Actual Netmask 255.255.255.0
Fieldbus Address 4
Fieldbus In/Out length 50 words
Fieldbus In/Out offset 0
RS-232 Software-Handshake Xon/Xoff
RS-422 point to point
CAN Cycle time 1 ms
CAN node address 0
CAN baudrate: 500 Kbit
Startup delay  0s
Display mode channel state
************************************
```

```
0: Exit menu
1: Change operating mode
2: Delete application data
3: Change CANbus configuration
4: Change Ethernet
5: Change in/out length for Fieldbus
6: Change offset for Fieldbus
8: CAN Cycle time
9: CAN node number
11: Change Fieldbus address
12: Change number of channels
13: Change time and date
14: Change RS-232 configuration
15: Change RS-422 configuration
16: Change startup delay
17: Change display mode
18: Change usage RS422 for PLC
Choice [Return; ESC exits menu]: 4
New IP address (192.168.0.6) ?
New Netmask (255.255.255.0) ?
```

The IP address can be changed without restart of the system. If the netmask is changed then a restart is necessary.

Serial

The CoDeSys programming tool connects itself locally per TCP/IP with the program PTerm. This program undertakes the data transfer to the control. Simultaneously also the other programmes as PMotion, PScope etc. can take hold of the control via the serial interface.

Important: The program PTerm must always be started and there must be a serial connection open to the control.

Since version 2.005:

Now it is possible to use the Pilz control panels of the PMI series directly in CoDeSys. Therefore the PMCprimo configuration command CD 18 must set to 1 in order the RS422 interface ist set for SoftPLC. The control panel can direcrtly access to variables of SoftPLC. It is defined an array of 1023 global variables and named with g_iModbusData. A modbus protocol is not necessary therefore. The boud rate is set to 38400 (8 bits, 1 stopbit and no parity). Motorola byteorder must be set.

If the RS422 interface is set for PLC also the programming must be done with this interface because the connection with Pterm is deactivated (localhost in CoDeSys) if CD18 is set to 1.

It is also possible to access the global data g_iModbusData from CAN. With SDO the index 0x3001 to 0x3008 it is possible to access 128 elements with sub index 0 to 127 (integer 16, 2 bytes) of the array. Therefore all 1024 array elements are accessible with SDO.

Since **version 2.006**:

The number of global variables g_iModbusData was increased from 1024 to 2048. The index for CAN is 3001 hex to 3010 hex and the subindex is 0 to 127.

12.2 Setting of a new connection

First of all a new entry is generated in the CoDeSys programming tools under the menu Online-> communication parameter and a name to be selected freely is allocated for this connection and then the corresponding list entry is modified.

The following parameter must be set:

Address:	With Ethernet the address of the control, with serial interface localhost or 127.0.0.1
Port:	1200 (Standard value)
Motorola Byteorder:	Yes (Standard value No), double click changes the value
Blocksize:	128 (Standard value)

Afterwards the connection to the control can be checked with the menu Online->Log in. The following window should appear then:

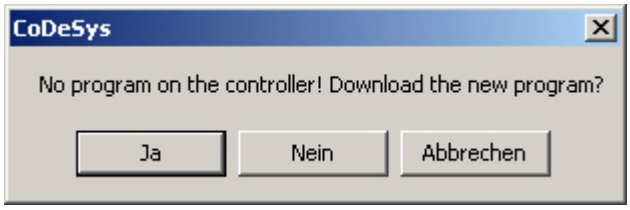


Illustration 4: Log-in in the control

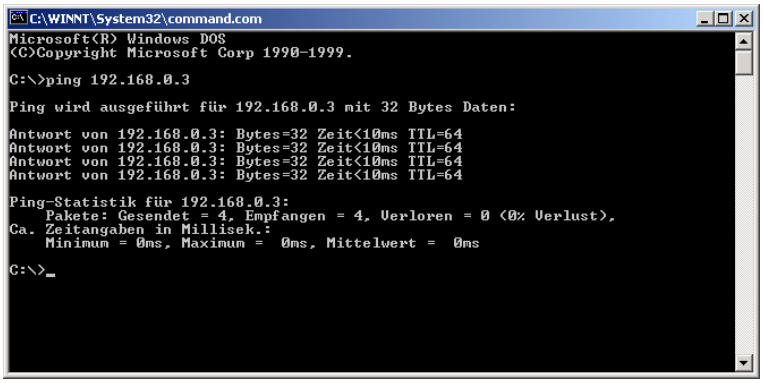
The window only appears when the module PLC_PRG includes an executable network and the project could successfully be compiled.

If this window is indicated it means that the communication could be built-up and the PLC-program can be downloaded.

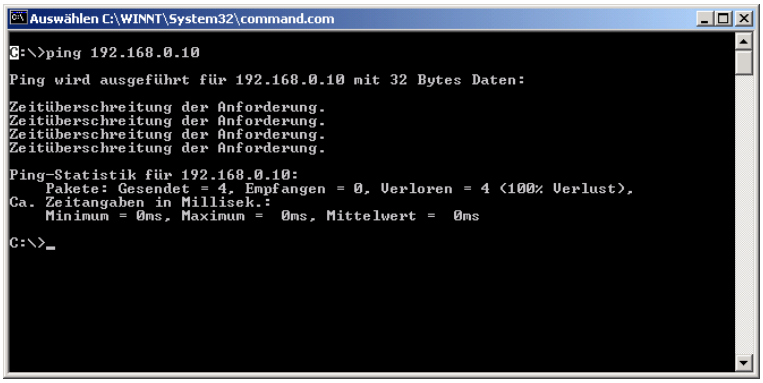
In case a failure message appears, it might be that the IP-address is not correctly configurated.

Remark: The following pictures are given from a german windows system!

First off all try the Command PING under the Start->Execute:

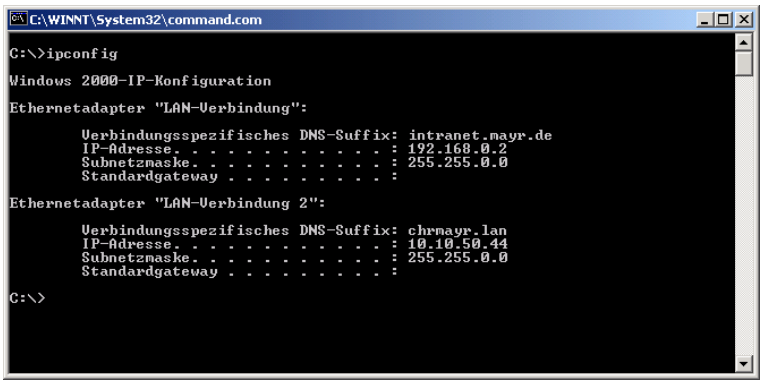


In this case a connection could be established, however, a IP-address may be used twice and a acknowledge message has been signalled back from a different PC.



If the issue appears in such a way then the control is either not connected, or the IP-address or the netmask are set incorrectly.

Check which IP-address and netmask the PC, on which the CoDeSys programming tools runs, has. This can be made under Windows NT or 2000 with the command IPCONFIG (Windows 95,98 and ME with the command WINIPCFG):



In this case two network cards are installed in the PC. The PMCprimo is connected at the first network card and can be set to a IP-address of 192.168.0.1 up to 192.168.255.255. The first two digits may not be different, as the netmask 255.255.0.0 does not allow this.

Please contact your system administrator in case of further questions.

13 SoftPLC in PMCprimo

The functionality of the SoftPLC is implemented in PMCprimo as independent task. The used preemptive multitasking operating system always selects this task, if tasks with a higher priority are not active. As the PLC must always be selected in a cyclic way and, therefore, this task is never in an idle running condition, this one has the second lowest priority. Only the task for the Motiongenerator (software option) has still a lower priority. As soon as the Motiongenerator must generate a curve, the PLC-task is not started with each cycle for 2 milliseconds. This means, the minimum cycle time has 2ms plus the operating time. If the Motiongenerator is not active, the PLC-task is restarted immediately. It depends on the size of the PLC-program which cycle time is completely to be realised.

Example: With PMCprimo 16+ 5000 IL-instructions needs approx. 0,5 ms -> When Motiongenerator active cycle time = 0,5 ms + 2 ms = 2,5 ms.
In case the Motiongenerator is not active, the cycle time amounts to 0,5 ms.

In case of this example the inputs are memorised and processed every 0,5 ms and then the outputs are updated.

This time can be extended if tasks with a higher priority, as for example the servicing of the program for PMCprimo-programs, require processor time. To be able to measure the cycle time the global variable g_diCycleTime has been implemented. This variable reproduces the cycle time without possible 2 ms pause, when the Motiongenerator is active.

13.1 Start of the system

During start of the system information as to the PLC-program are indicated.

The information of the project stored last is indicated which generate in CoDeSys with the function Bootproject and secured in Flash.

```
Ser.No.:441, Version 1.007beta6 Feb 27 2002, 15:32:49
Operate Mode: STANDALONE
MOTION INSTALLED
SOFTPLC INSTALLED
*** Date of Project: 2002-03-13
*** Project: Sample.pro
*** Title: Sampleproject
*** Version: 1.0
*** Author: Max Mustermann
*** Description: Sampleproject for documentation
Channel 0.1 found
Channel 0.2 found
Channel 0.3 found
RESTORING DATA...
0.1:
0.2:
0.3:
0.1:
READY RESTORING DATA
0.1:
```

13.2 Non transient storage

Completely there are 256KB of a non transient flash storage for programs available. This storage is used for half of PMCprimo-programs and PLC-programs each. Therefore the PLC-program can be 128KB big the maximum. As of Firmware version 1.008c with PMCprimo command “CD” the available storage for PLC-programs can be set. For detailed information please refer to the PMCprimo reference manual.

Example:

```
0.1: CD
0.1:
Actual configuration:
Operate Mode: STANDALONE
Actual IP address 10.10.180.100
Actual Netmask 255.255.0.0
Fieldbus In/Out length 50 words
Fieldbus In/Out offset 0
Channel 5 disabled
RS-232 Software-Handshake Xon/Xoff
CAN Cycle time 2 ms
CAN node address 0
Flashmemory for SoftPLC 128KB
Startup delay  0s
************************************
0: Exit menu
1: Change operating mode
2: Delete application data
3: Change CANbus configuration
4: Change Ethernet
5: Change in/out length for Fieldbus
6: Change offset for Fieldbus
7: Enable channel 5 (CAN-encoder)
8: CAN Cycle time
9: CAN node number
10: Flashmemory for SoftPLC
14: Change RS-232 configuration
16: Change startup delay
Choice [Return; ESC exits menu]: 10
Flashmemory for SoftPLC (0: 128KB, 1:192KB 2:256KB) 128KB ? 192
All data will be deleted ! Are you sure? Y/N: Y
Restore in Flash necessary!
Please exit menu with ESC to save parameters and then reboot system !
```

13.3 Working storage

The available RAM-storage is dynamically distributed among all tasks. 16MB-storage are sufficient for all thinkable programs.

13.4 Battery back-up storage

PMCprimo Drive:

No battery back-up available.

PMCprimo 2+2:

In case of the PMCprimo 2+2 7392 Bytes are available for the PLC as battery back-up storage. If the version is less then 2.004 then 2048 bytes are available. This storage can take up variables which are marked with the keyword RETAIN. The computer automatically files these variables into this special storage range

PMCprimo 16+:

In case of the PMCprimo 2+2 124 KBytes are available for the PLC as battery back-up storage. This storage can take up variables which are marked with the keyword RETAIN. The computer automatically files these variables into this special storage range

PMCprimo Drive2:

If the optional expansion board is mounted then 7382 Bytes are available.

13.5 Integration of the SoftPLC in PMCprimo

The installed function library adds function blocks to the SoftPLC. Independent actions can be started in PMCprimo with these function blocks. PMCprimo undertakes this action and independently carries them out, without further function calls of PLC.

If for example a positioning shall be started the accelerations and speed can be set with one module each and then the positioning can be started with the function block MoveToAbsolutPosition. PMCprimo independently undertakes now the complete setting value calculation and positioning control. It is not necessary for PLC to do anything else for the positioning. Each function block has an output bDone with which the complete performance is re-signalled. This means for example with MoveToAbsolutePosition that the target position was achieved and the axis has braked again to the speed O.

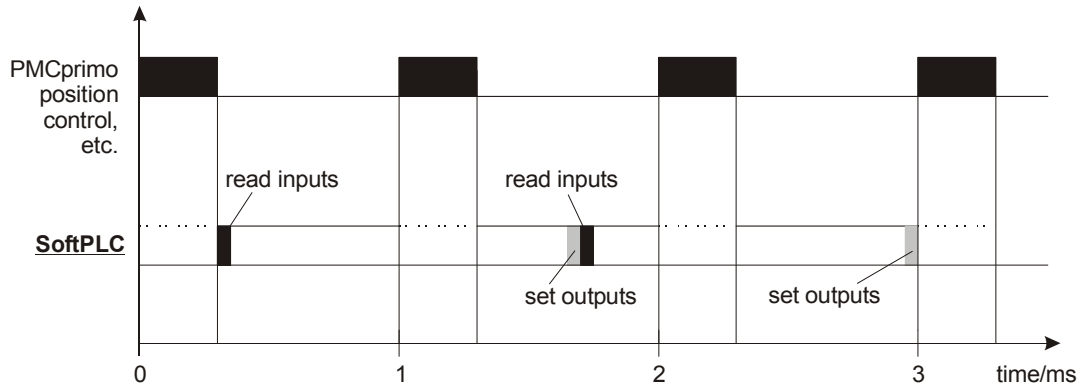


Illustration 5: time response SoftPLC

The PMCprimo-task is executed in every millisecond and undertakes the setting value calculation, positioning control etc. In case the setting value calculation requires a longer working time, the PLC-Task can be interrupted several times.

When starting the PLC-program the inputs are registered and then the actual PLC-program is executed. At the end the outputs are updated and the cycle starts again.

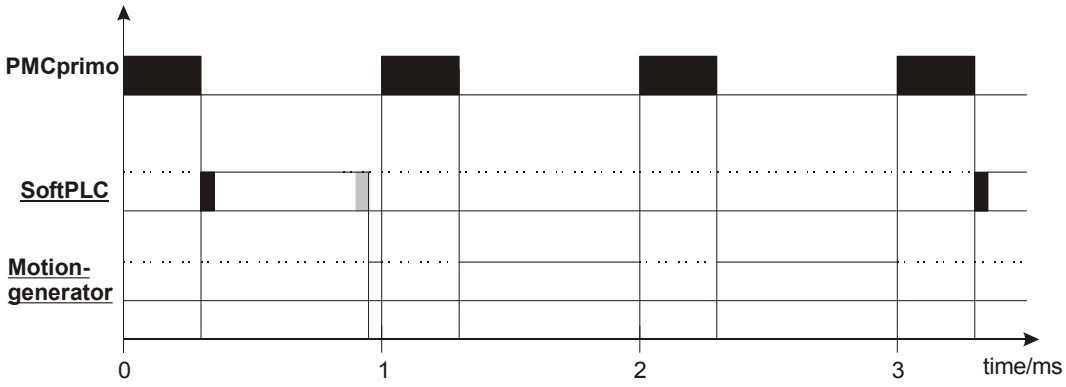


Illustration 6: time response SoftPLC with Motiongenerator

Is additionally the Motiongenerator active, the SoftPLC carries out its standard calculation cycle and after the outputs have been updated, the SoftPLC stops for 2 milliseconds enabling the processor to carry out the Task for the Motiongenerator. As soon as the curve has been calculated, the Task for the Motiongenerator is finished and there is no pause.

As of **version 2.006** defintion of new tasks:

Now it is possible to add 4 tasks inside the SoftPLC with high priority. For this a new file taskkonfig.xml is necessary.

In the task configuration the type triggered by an external event is available. The cycle time is set with the input in the cyclic type. First the type has to be set to cyclic and the time typed in and then the type has to be changed to triggered by an external event. Then the task TASK_T01 can be chosen.

The four tasks are executed with a higher priority directly in the system. The normal PLC task is therefore interrupted and it is possible to react faster on some inputs.

Important:

Because the fast tasks can interrupt the normal PLC_PRG task care must be taken with the outputs.

13.6 PMCprimo-programs

A complete application with the SoftPLC is possible, however, the processing is completely carried out in the Host, i.e. every moving command is started in the Host.

PMCprimo-programs are possible as up to now. These programs can be started by the PLC and be processed in a parallel way. Here the distribution Host and nodes remains existing. Herewith a faster processing with the change of an input is possible.

There are two possibilities in order to start PMCprimo programs from the PLC:

Access to the bus variables: These variables can be defined as trigger variables under PMCprimo and then start a pre-selected program.

A program can directly be started via a function block. For that the program name must be indicated in the PLC.

Example for a PMCprimo-program:

```
ES MOVE
CH1.1;SV100000;SA500000;DC500000;MA10000
NS
```

This program is locally processed on the node 1. The program can directly be started from an input on this node (for example with the command DI1.1:2+,MOVE). In this case the movement is carried out with a positive impulse of the input 2, without loading the PLC. It is processed within a millisecond as the performance is carried out directly on the node.

The program can be carried out with the function block ExecuteSequence(´Move´) by the PLC. However, the processing is more slowly. The time is determined by the communication speed and size of the PLC-program.

14 Function library primo.lib

Notes for the function library primo.lib:

The PLC-Function blocks call the corresponding PMCprimo-commands.

Every function block has an input bExecute. The corresponding action is started with an rising edge from 0 to 1 in PMCprimo. Is the input bExecute on 0, the function block is relocated.

Every function block has three output variables.

bDone:	Signals True back, if the action is completed. False, if the action is still under operation.
bError:	If an error has occurred , TRUE is signalled back here.
iErrorNumber:	Should bError be TRUE, then an error number is signalled back here which specifies the error in details.

The PMCprimo function library is divided in several sub-files. This subdivision only serves for clearness and does not have any effect on the integration in a PLC-program.

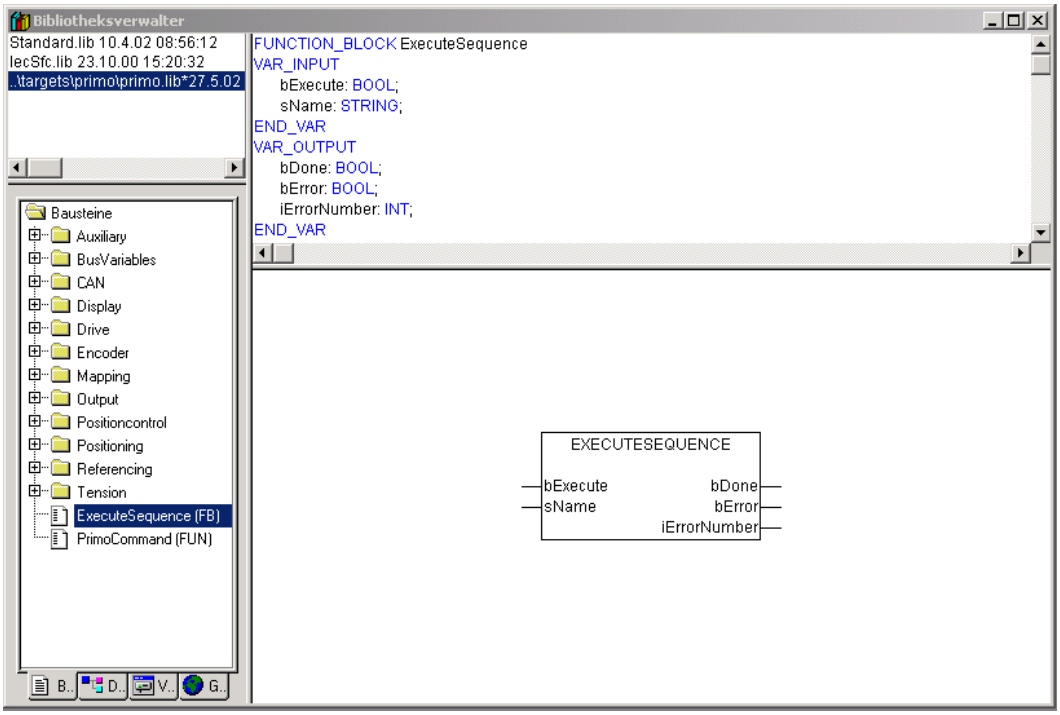


Illustration 7: PMCprimo Function library

The input and output variables are adapted to the Hungarian notation. As a result an easy connection between variable names and value ranges is possible.

Präfix	Meaning	Value range
b	bool	True or False
di	double int	- 2147483648 up to 2147483647
i	int	-32768 up to 32767
pdi	pointer double int	Pointer to double int
s	string	
udi	unsigned double int	0 up to 4294967295
ui	unsigned int	0 up to 65535
usi	unsigned short int	0 up to 255

14.1 Versions of Primo.lib

In the directory Target/Primo different libraries are stored. Depending to the firmware version a corresponding version of library has to be used. If the wrong version is used then new function can't be used. To use the right version either rename or copy the file to primo.lib or use the library manager in CoDeSys to replace library primo.lib by the right version.

Released versions:

Primo_V1_008.lib: For firmware version 1.008 and also for 1.009; first version

Primo_V1_010.lib: For firmware version 1.010; new function blocks

With this versions the maximum numbers of POU is 512.

The version 1.008 can also used with firmware version 1.010 if the new function blocks are not needed. It is not possible to use the library 1.010 with the firmware version 1.008 or 1.009 because there will be error messages while downloading the project.

Primo_V2_000.lib For firmware version 2.000 up to 2.002; new function blocks and some changes

Primo_V2_003.lib For firmware version 2.003

Primo_V2_004.lib For firmware version 2.004

Use this version if the firmware version 2.004 is in PMCprimo. Don't use another library.

From version 2.000 the maximum numbers of POUs is 4096. If there is an error message while compiling the project then change in the target setting the maximum number of POUs.

14.2 Example of a program

In the following example there is a program which calls the function block MoveToAbsolutePosition.

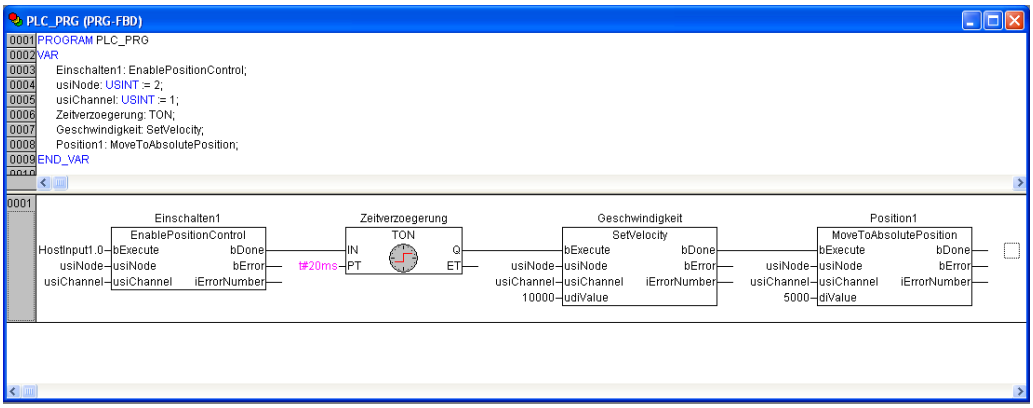


Illustration 8: Example of a function block

The program switches the axis 1 to the node in a position control as soon as the input 1 is energised with 24V. Afterwards a time delay of 20 ms runs enabling the servo control to overtake this information and to build up the torque. Afterwards the demand speed is set and an absolute positioning is executed. When this movement is completed the output bDone of the function block position 1 TRUE is resignalled.

14.3 Global variables

With version 2.004 additional global variables are available.

- g_iModbusData [0..1023] OF INT:

  With the array it is possible to access directly 1024 variables with the Modbus protocol. It is not necessary to call the function blocks SetBusVariable and GetBusVariable for this variables. The variables can be accessed from address 1000 to 2034 at the Modbus side.

- g_PrimoData
  It is possible to add the elements PMCprimo Global Variables in the PLC configuration module PMCprimo. Here it is possible to access different data without using the corresponding function blocks.
  Then the global array g_PrimoData[1..20] can be used.

- STRUCT
  diDemandPosition: DINT;
  diActualPosition: DINT
  diFollowingError: DINT;
  diActualSpeed: DINT;
  iActualError: INT;
  iActualState: INT;
  diReserved:ARRAY [0..2] OF DINT;
  END_STRUCT

Note: to keep the CAN bus load low only data from the host is available. Informations from the node are not stored in this global structure. Therefore the function blocks (for example DisplayActualPosition) has to be used.

Change as of version 2.006:

- Now the array g_iModbusData accepts 2048 (0…2047) values. The variables can be accessed from address 1000 to 3047 at the Modbus side.

14.4 The subdivision

The function blocks are listed in table of contents to explain the affiliation.

Auxiliary:

Funtion modules for definition and setting of the analogue auxiliary output.

BusVariables:

Function blocks for writing and reading of the bus variables to PMCprimo

CAN:

Function blocks for writing and reading of SDO-data. As a result an access to the object list of a CANopen unit in the network is possible. Furthermore a PMCtendo DD4 can be defined as PMCprimo-axis.

Display:

Function blocks for a feedback of demand and actual positions and other values from the control.

Drive:

A function block for communication with the cascade PMCtendo DD4 with PMCprimo Drive.

Encoder:

Function blocks for setting the measuring system. Here the kind (incremental SSI or similar) and the resolution can be set.

Mapping:

Function blocks for the Master-/ Slave – relations with the appropriate parameters.

Output:

Function blocks for definition of outputs which are directly operated in a 1 ms cycle by PMCprimo. These are for example cams, error outputs, reference outputs etc.

Positioncontrol:

Function blocks for setting of the position control. Here the basic pre-settings are possible, as for example contouring error limits or position control parameters.

Positioning:

Function blocks for easy moving commands. Here speeds, accelerations and targets are given.

Referencing:

Function blocks for the cyclic reference correction. The reference error correction is also executed in the background without using PLC. This kind of operation is activated and parameterised by calls.

Tension:

Function blocks for setting of the web tension control. There the web tension control can be activated. Furthermore the control parameter can be changed for it.

Wait:

Function blocks for special wait functions. PMCprimo waits for example for a defined position before bDone goes TRUE. Therefore the critical time polling from the PLC is avoided.

14.5 Auxiliary

The analogue outputs (+/-10V) can be defined as auxiliary outputs with the function blocks DefineAuxiliaryOutput, SetMonitorOutputFunction, SetMonitorOutputGain and SetMonitorOutputOffset. These auxiliary outputs can be used in order to connect an oscilloscope or multimeter for example and to indicate various values from the control.

However, a frequency converter can also be connected and the speed can be controlled with the function SetMonitorOutputOffset without position control. As a result a slowly control could be realised, i.e. dependent on an analogue input for example the frequency converter can be controlled faster or more slowly.

The following graphics shows the connections of the various functions.

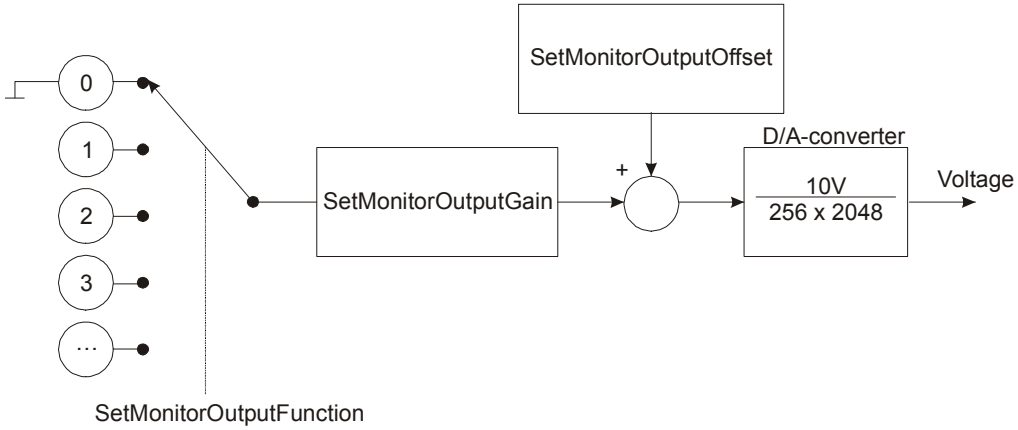


Illustration 9: Analogue auxiliary output

DefineAuxiliaryOutput

PMCprimo-command: AO (Set **a**uxiliary **o**utput channel)

Function list: **Auxiliary**

Description:
This command allocates the monitor signal for the current channel to one of the analogue outputs on the same board. The monitor function SF may be defined for the current channel at any time. The AO command is not available on different nodes. The auxiliary output channel may be set for a particular analogue output only when its channel is in the motor off or virtual motor modes. If the channel is in any other state then the analogue output is not available for use as a monitor output. Conversely, if the analogue output has been allocated to a channel as a monitor signal, then this channel cannot be taken out of motor off or virtual mode.

To return analogue output channel to normal operation, use AO0 on the channel where the auxiliary output is defined. Note that it is also possible to have the auxiliary output signal allocated to the current channel when it is in virtual mode; the signal does not have to be defined on a different channel's output. This may be useful in open-loop control applications.

In PMCprimo Drive the command AO4 enables a limitation of the torque. PMCtendo Version 3.55 is required and the parameter DILIM must be set to 1 (after this change it must done SAVE and COLDSTART). The value 3280 is equal to the IPEAK value.

Input variables:

bExecute (BOOL)	In case of a change from 0 to 1 the function is activated 0 sets the function block back and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiOutput (USINT)	The output number from 1 to 8, 0 cancels the definition.

Output variables:

bDone (BOOL)	The function has been executed (True) or is under operation (False)
bError (BOOL)	True: an error has occurred, False: no error
iErrorNumber (INT)	Indicates the exact reason of error (see GetError page 203)

Connected functions:

SetMonitorOutputFunction, SetMonitorOutputGain and SetMonitorOutputOffset

Factory setting: No output defined

Examples:

Declaration:

```
INST: DefineAuxiliaryOutput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

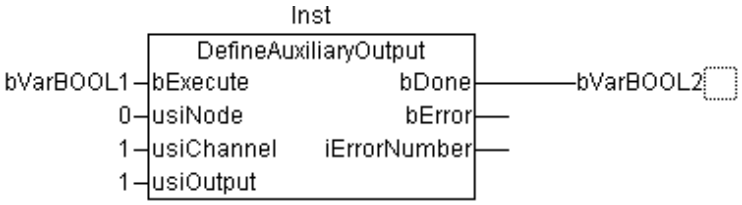
Example in IL:

```
CAL Inst(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiOutput:=1)
LD Inst.bDone
ST bVarBOOL2
```

Example in ST:

```
Inst((bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiOutput:=1);
bVarBool2:= Inst.bDone
```

Example in FBD:



SetMonitorOutputFunction

PMCprimo-command: SF (**S**et monitor output **f**unction)

Function list: **Auxiliary**

Description:

The output function analogue auxiliary output can be set with this function block. It facilitates for example the setting of the control parameter of the corresponding axis. For the output of the analogue values an analogue output momentary not used is taken with the DefineAuxiliaryOutput (page 35).

Arguments:

bExecute (BOOL)	Setting is executed with a change from 0 to 1. 0 sets the function block back and the output variables are set to False or 0.
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiValue (USINT)	The number of the function

Value	Function
0	no function
1	demand speed
2	actual speed
3	contouring error
4	speed error
5	absolute demand position
6	absolute actual position
7	average actual speed
8	speed master axis
9	average speed master axis
10	demand value web/torque control
11	reference error
12	encoder moment receiver
13	demand speed including reference error correction
14	Velocity ratio of master/slave. Therefore the average velocities (VT page 65 and BT page 121) are used. The ratio is calculated as follow: Output = DV * KM / (DV master) + OM

Output variables:

bDone (BOOL)	The function was carried out (True) or is still under operation (False)
bError (BOOL)	True: an error has occurred; False: No error
iErrorNumber (INT)	Indicates the exact error reason (see GetError page 203)

Connected functions:

DefineAuxiliaryOutput, SetMonitorOutputGain and SetMonitorOutputOffset

Factory setting: 0 (no function)

Examples:

Declaration:

INST: SetMonitorOutputFunction;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;

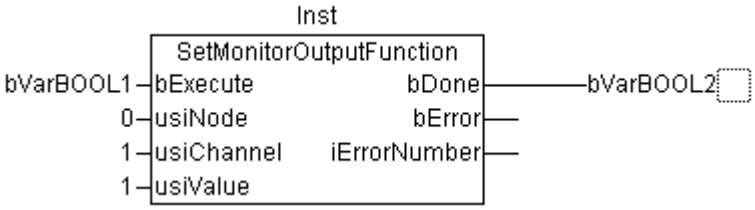
Example in IL:

CAL Inst(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiValue:=1)
LD Inst.bDone
ST bVarBOOL2

Example in ST:

Inst((bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiValue:=1);
bVarBool2:= Inst.bDone

Example in FBD:



SetMonitorOutputGain

PMCprimo-command: KM (Set **m**onitor output gain)

Function library: **Auxiliary**

Description:

The analogue auxiliary output is not influenced by the factors of the control algorithm. The analogue auxiliary output depends not on the pre-setting of the control parameter (SetControlWord page 182). The value to be given at the analogue auxiliary output is multiplied with the set value before its output. The output signal can be inverted by changing the preceding sign.

Input variables:

bExecute (BOOL)	The setting is executed with a change from 0 to 1 0 sets the function block back and the output variables are set to False or 0.
usiNode (USINT)	The node number in the linked system , 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
iValue (INT)	The amplification factor.

As of **Primo_V2_006.lib**: diValue (DINT)

Output variables:

bDone (BOOL)	The function was executed (True) or is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

DefineAuxiliaryOutput, SetMonitorOutputFunction and SetMonitorOutputOffset

Factory setting: 1

Example:

Declaration:

```
INST: SetMonitorOutputGain;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

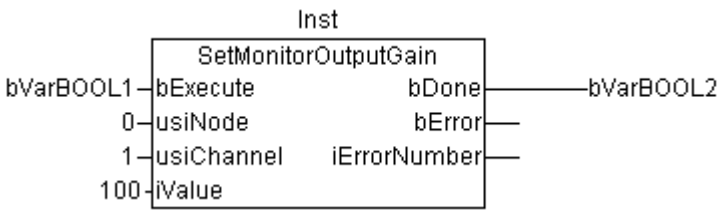
Example in IL:

```
CAL Inst(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, iValue:=100)
LD Inst.bDone
ST bVarBOOL2
```

Example in ST:

```
Inst(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, iValue:=100);
bVarBool2:= Inst.bDone
```

Example in FBD:



SetMonitorOutputOffset

PMCprimo-command: OM (Set **m**onitor **o**utput offset)

Function : **Auxiliary**

Description:

The analogue auxiliary output can be provided with a firm voltage level (Offset) with this function block.

Offset in mV = 4,88 mV * SetMonitor OutputOffset

Input variables:

bExecute (BOOL)	In case of a change from 0 to 1 the setting is executed 0 places the function block back and the output variables are set to False or 0.
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
iValue (INT)	The voltage level (Offset).

Output variables::

bDone (BOOL)	The function was executed (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

DefineAuxiliaryOutput, SetMonitorOutputFunction and SetMonitorOutputGain

Factory setting:: 0

Example:

Declaration:

```
INST: SetMonitorOutputOffset;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

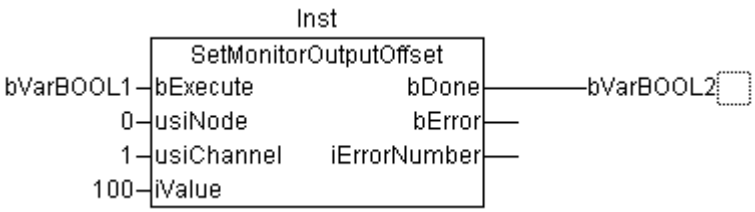
Example in IL:

```
CAL Inst(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, iValue:=100)
LD Inst.bDone
ST bVarBOOL2
```

Example in ST:

```
Inst(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, iValue:=100);
bVarBool2:= Inst.bDone
```

Example in FBD:



14.6 BusVariables

The BusVariables serve as interface to external units as for example, control units. The variables are set via Modbus, CAN, Profibus-DP Slave and Ethernet-interface. The various Bussystems take all hold of the same variables. If several Bussystems are used simultaneously, the last transmission decides which value the variable finally has.

GetBusVariable

PMCprimo-command: $B1 .. $B108

Function library: **BusVariables**

Description:

With this function block the actual value of a BusVariable can be read in the PLC.

Input variables:

bExecute (BOOL)	The variable is set in case of a change from 0 to 1 0 places the function block back and the output variables are set to False or 0 except diValue
iNummer (INT)	The number of the variable from 1 to 108.

Output variables::

bDone (BOOL)	The value was read (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	The actual value of the BusVariable.

Connected functions:

SetBusVariable

Examples:

Declaration:

```
INST: GetBusVariable;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
diValue: DINT;
```

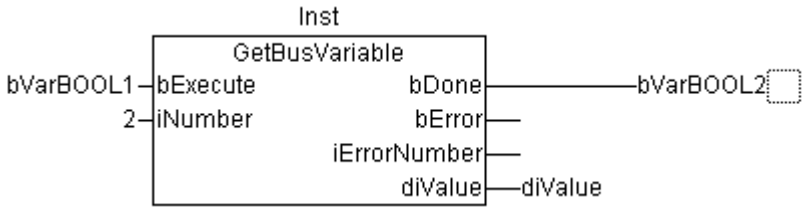
Example in IL:

```
        CALINST(bExecute := bVarBOOL1, iNumber := 2)
        LD INST.bDone
        ST bVarBOOL2
        EQ TRUE
        NOT
        JMPC  else1_0
        LD INST.diValue
        ST diValue
else1_0:
```

Example in ST:

```
Inst(bExecute:= bVarBOOL1, iNumber:=2);
bVarBool2:=Inst.bDone;
IF bVarBool2 = TRUE THEN
        diValue:=Inst.diValue;
END_IF
```

Example in FBD:



SetBusVariable

PMCprimo-command: $B1 .. $B108 = value

Function library: **BusVariables**

Description:

The PLC can transmit a value to the BusVariable with this function block.

This BusVariable can be defined as TriggerVariable and is, therefore, able to call a PMCprimo-program.

Input variables:

bExecute (BOOL)	The setting is executed with a change from 0 to 1. 0 places the function block back and the output variables are set to False or 0.
iNumber (INT)	The number of the variable from 1 to 108 The variables 101 to 108 are linked to virtual inputs in PMCprimo
diValue (DINT)	The new value of the variable

Output variables:

bDone (BOOL)	The function was executed (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Arguments:

Connected functions:

GetBusVariable

Examples:

Declaration:

```
INST: SetBusVariable;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

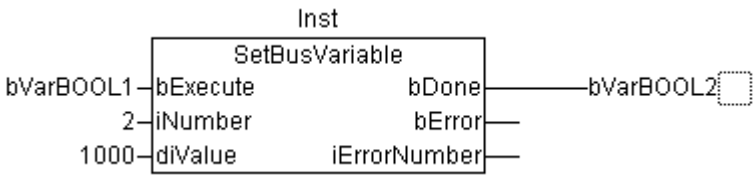
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, iNumber := 2, diValue:=1000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
Inst(bExecute:= bVarBOOL1, iNumber:=2, diValue:=1000);
bVarBool2:=Inst.bDone;
```

Example in FBD:

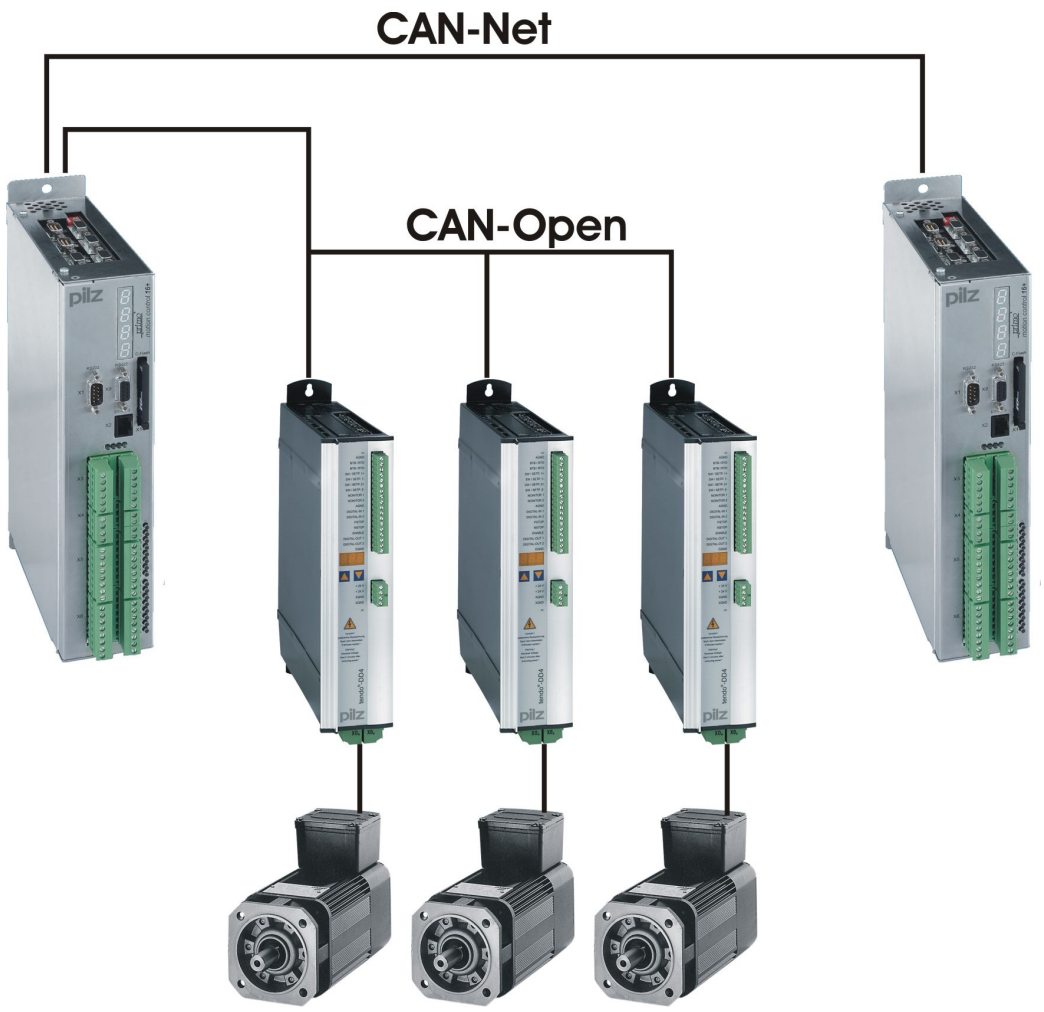


14.7 CAN-Net and CAN-open

As of **version 2.000** multiple CAN networks per device are supported by PMCprimo.

The CAN net, in which all PMCprimo devices are connected with each other is called “**CAN-Net**”.

The CAN net which connects a PMCprimo device with some servo drives (PMCtendo DD4) is called “**CAN-Open**”.



CANPositionControlDrive and CANNetworkPositionControlDrive

PMCprimo-command: PD (**p**ositioncontrol to **d**rive)

Function library: **CAN**

Description:

With this function block one or two PMCtendo DD4 with speed demand values can be controlled via the CANbus in the VM0-Modus on the axis 2 and 3 (PMCprimo Drive) or four PMCtendo DD4 (PMCprimo 2+2).

The position is read from the encoder input or also via the CANbus. It can be set with the function SetFeedbackEncoder (page 76).

The transmission of the speed demand value is made in a 4 ms step. If the actual position is also transmitted via the CAN, the actual position is transmitted within this time. An interpolation is made in the system in 1ms steps. These intermediate values are used for possible slave axes.

A PMCprimo Drive can still control 2 further axes and, therefore, a 3 axis system arises. The position control parameters are set as with the internal axis and all movement functions are also possible.

The only difference to an internal axis is the reference function. The reference functionality requires an utmost possible exact value. If the actual position is transmitted via CAN, an exact position measuring is not possible. If the reference functionality is required with an axis via CAN, either an additional encoder line (from the encoder simulation of the servo control to the free encoder input) must be arranged or the determined reference position has a temporary inaccuracy. As only one additional encoder input is available, of course, only one axis can be used for this exact position determination.

The definition is cancelled with the value 0 and the connection is disconnected via CAN.

Enhancement as of version 2.000:

The function block CANNetworkPositionControlDrive was implemented addionally, to access the second can controller of a PMCprimo 16+ or a can expansion card. For that, another input variable parameter usiCanNetwork was defined, which differs between the CAN-net and CAN-open.

Enhancement as of version 2.000:

On the PMCprimo 16+ the command can be used on a maximum of 16 axes. The cycle time for the CAN bus transmission is depending on the number of PMCtendo DD4 in the CAN net.

With value 0 the definition is cancelled and the connection via CAN is disconnected.

	Usabe axis numbers	Number per CAN
PMCprimo Drive	2+3	2 just CAN
PMCprimo Drive2	2-10	(4[*1]) 8[*2] per CAN-Netz
PMCprimo 2+2	1-4	4
PMCprimo 16+	1-16	(4[*1]) 8[*2] per CAN-Netz

([*1] till version 2.004) ([*2] as of version 2.005)

Cycle time:

In the main network, mode standalone, the cycle time can be adjusted with the „**CD**“ command. In mode “Host+Node” the cycle time is always 4 ms. In an extended CAN network the cycle time is always choosen automatically, according to the following table:

CANPositionControlDrive (PD)	500 KBit	1 MBit
1-2	1 ms	1 ms
3-4	2 ms	1 ms
5-8	4 ms	2 ms

Input variables:

bExecute (BOOL)	The setting is executed with a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiCanNode (USINT)	The CAN node address of the PMCtendo DD4, 0 stops the connection

Output variables:

bDone (BOOL)	The function was executed (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetFeedbackEncoder

Factory setting: 0 (no connection)

With **version 2.003** a new function block CANNetworkPositionControlDrive is available. This block has an additional input for choosing the CAN network with PMCprimo 2+2, 16+ and PMCprimo Drive2.

Additional input variable:

usiCanNetwork	0: CAN main network. 1: CAN at expansion board or second CAN controller

Example:

Declaration:

```
INST: CANPositionControlDrive;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

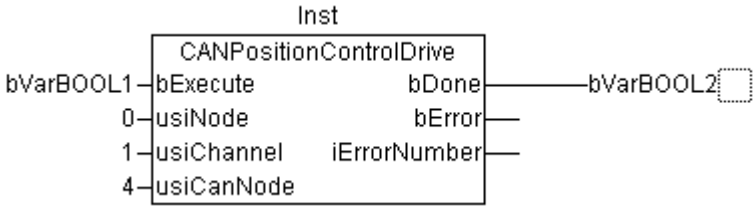
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChanel:= 1, usiCanNode :=4)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
Inst(bExecute := bVarBOOL1, usiNode := 0, usiChanel:= 1, usiCanNode :=4);
bVarBool2:=Inst.bDone;
```

Example in FBD:



GetCanSDO

PMCprimo-command: QR (Read SDO)

Function library: **CAN**

Description:

With this function block PMCprimo can take hold of the object list of a CAN unit per SDO and is, therefore, able to read actual values or configurate the unit, for example. The configuration of the parameter must be taken from the user manual of the corresponding network participant. The read value can be allocated to a variable.

Input variables:

bExecute (BOOL)	The reading function is started with a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiCanNetwork	As of **version 2.000**: 0: CAN main network. 1: CAN at expansion board or second CAN controller
usiCanNode (USINT)	The CAN node address CANopen unit
uiIndex (UINT)	The index of the object library
usiSubIndex (USINT)	The sub index for the access of the object library
usiBytes (USINT)	The number of the Bytes (1 to 4).

Note: The number of the Bytes must coincide with object library of the CAN unit.

Output variables:

bDone (BOOL)	The value has been read (True) or is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	The read value

Connected functions:

SetCanSDO

Examples:

Declaration:

```
INST: GetCanSDO;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

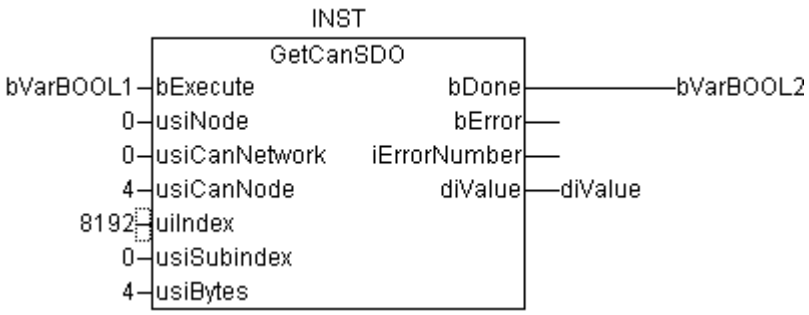
Example in IL:

```
CAL INST(bExecute := bVarBOOL1, usiCanNode := 4, uiIndex := 16#2000, usiSubindex := 0,
usiBytes := 4)
LD INST.bDone
ST bVarBOOL2
ST diValue
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiCanNode:= 4, uiIndex:= 16#2000, usiSubindex:= 0, usiBytes:=
4 );
bVarBool2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



SetCanSDO

PMCprimo-command: QS (Send SDO)

Function library: **CAN**

Description:

PMCprimo can have access to object library of a CAN-unit per SDO and write for example demand values or configurate the unit. The configuration of the parameter must be taken from the user manual of the corresponding network station.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set from False to 0
usiCanNetwork	As of **version 2.000**: 0: CAN main network. 1: CAN at expansion board or second CAN controller
usiCanNode (USINT)	The CAN node address CANopen unit
uiIndex (UINT)	The index of the object library
usiSubIndex (USINT)	The sub-index for the access to the object library
usiBytes (USINT)	The number of the Bytes (1 to 4).
diValue (DINT)	The value to be written.

Note: The number of the Bytes must coincide with the object library of the CAN-unit.

Output variables:

bDone (BOOL)	Value was written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

GetCanSDO

Examples:

Declaration:

```
INST: SetCanSDO;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

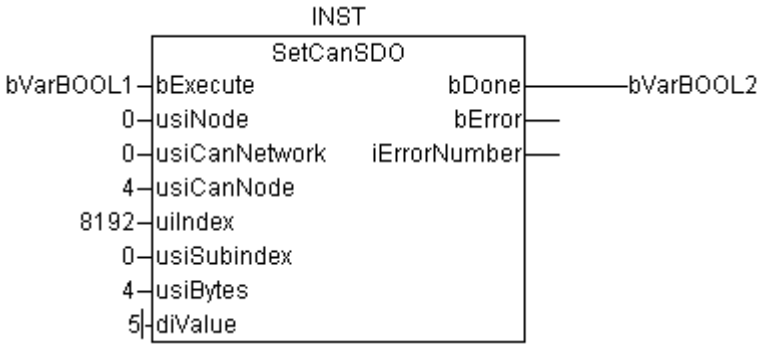
Example in IL:

```
CAL INST(bExecute := bVarBOOL1, usiCanNode := 4, uiIndex := 8192, usiSubindex := 0,
usiBytes := 4, diValue := 5)
LD INST.bDone
ST bVarBOOL2
ST diValue
```

Example in ST:

```
Inst(bExecute:= bVarBOOL1, usiCanNode:= 4, uiIndex:= 16#2000, usiSubindex:= 0, usiBytes:= 4,
diValue:=5);
bVarBool2:=Inst.bDone;
```

Example in FBD:



14.8 Display

Various values from the control can be read into the SoftPLC with the display function and further processed there.

DisplayActualPosition

PMCprimo-command: DP (**D**isplay actual **p**osition)

Function library: **Display**

Description:

The actual position of the selected axis is read in a variable with this function block. The unit of the actual position is increment.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been read (True) or is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact the error cause (see GetError page203))
diValue (DINT)	The actual position

Also refer to:

DisplayDemandPosition und DisplayFollowingError

Examples:

Declaration:

```
INST: DisplayActualPosition;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

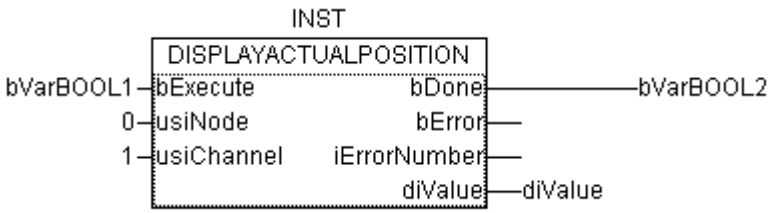
Examplel in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1);
bVarBool2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



DisplayAnalogueInput

PMCprimo-command: DA (**D**isplay **a**nalogue input)

Function library: **Display**

Description:

The current value of the analogue input is indicated with this function block. The value range is ±2047. This corresponds to +/- 10V.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been read (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203))
diValue (DINT)	The actual value of the analogue input.

Examples:

Declaration:

```
INST: DisplayAnalogueInput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

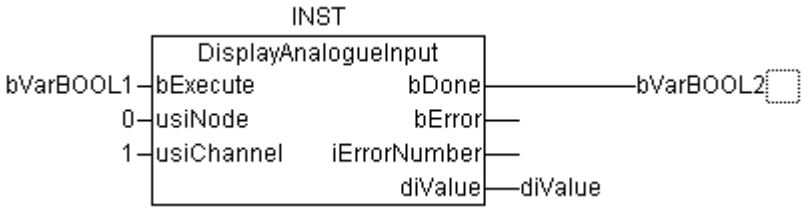
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1);
bVarBool2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



DisplayDemandPosition

PMCprimo-command: DD (**D**isplay **d**emand position)

Function library: **Display**

Description:

The actual demand position of the selected axis is read into a variable with this function block. The unit of the demand position is increments.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been read (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203))
diValue (DINT)	The actual demand position.

Also refer to:

DisplayActualPosition und DisplayFollowingError

Example:

Declaration:

```
INST: DisplayDemandPosition;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

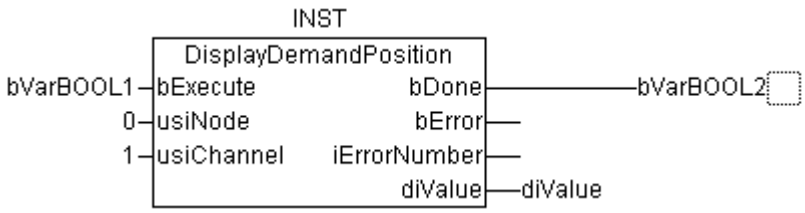
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1);
bVarBool2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



DisplayFollowingError

PMCprimo-command: FE (Display **f**ollowing **e**rror)

Function library: **Display**

Description:

The actual following error of the selected axis is read in a variable with this function block. The following error is the difference between demand position and actual position. The unit is increments.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been read (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	The actual following error.

Also refer to:

DisplayActualPosition und DisplayDemandPosition

Examples:

Declaration:

```
INST: DisplayFollowingError;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

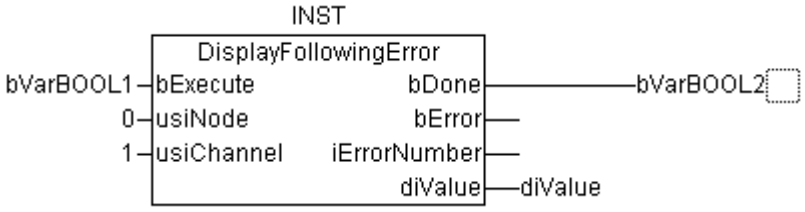
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1);
bVarBool2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



DisplayPositionOverflowCounter

PMCprimo-command: BC (Set position overflow counter)

Function library: **Display**

Description:

The actual position overflow counter of the selected axis is read in a variable with this function block. The unit of the actual position is increment.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been read (True) or is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact the error cause (see GetError page203))
diValue (DINT)	The actual value of position overflow counter

Also refer to:

DisplayDemandPosition und DisplayFollowingError

Examples:

Declaration:

```
INST: DisplayActualPosition;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

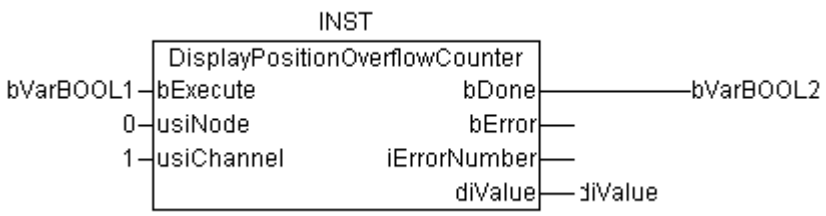
Example! in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1);
bVarBool2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



DisplayPositionBound

PMCprimo-command: SB (**S**et **p**osition bound)

Function library: **Display**

Description:

The actual position bound of the selected axis is read in a variable with this function block. The unit of the actual position is increment.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been read (True) or is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact the error cause (see GetError page203))
diValue (DINT)	The actual position bound

Also refer to:

DisplayDemandPosition und DisplayFollowingError

Examples:

Declaration:

```
INST: DisplayActualPosition;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

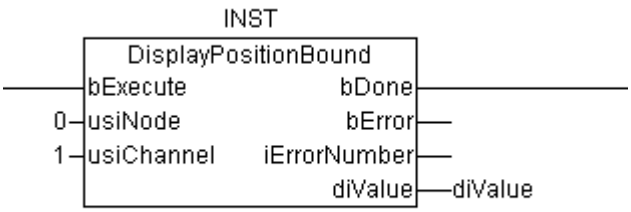
Example1 in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1);
bVarBool2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



DisplayReferenceError

PMCprimo-command: DF (**D**isplay re**f**erence error)

Function library: **Display**

Description:

The last reference error of the selected axis is read in a variable with this function block.The unit is increments.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	The value has been read (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	The last reference error.

Also refer to:

DisplayActualPosition and DisplayDemandPosition

Examples:

Declaration:

```
INST: DisplayReferenceError;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

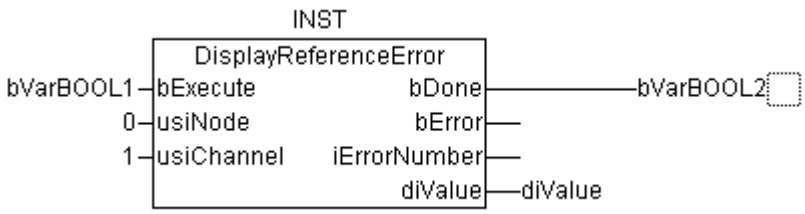
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1);
bVarBool2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



DisplayReferenceLengthFalse

PMCprimo-command: ZX (Display length reference signal false)

Function library: **Display**

Description:

The length measured before is read-out with this function block, over which the reference signal had the condition False. This indication is only made when Bit 0 of SetReferenceFilterOptionWord (page 267) is set to 1. In case Bit 0 of the SetReferenceFilterOptionWord is set to 0, then automatically ZX is also set to zero.

Note:

To be able to measure the length of the reference signal, the edge to be recognised is always changed internally. The time for the reprogramming is the debouncing time (Function SetReferenceHoldoffTime page 269). Signals which are shorter than the debouncing time or 1 millisecond cannot be measured and result a wrong value.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been read (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	The length of the reference signal.

Also refer to:

DisplayReferenceLengthTrue

Examples:

Declaration:

```
INST: DisplayReferenceLengthFalse;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

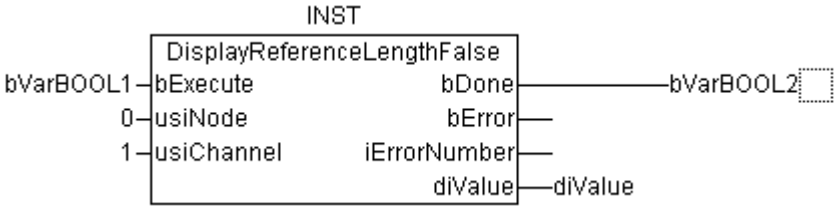
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1);
bVarBool2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



DisplayReferenceLengthTrue

PMCprimo-command: ZY (Display length reference signal true)

Function library: **Display**

Description:

The length measured before is read with this function block, over which the reference signal had the condition TRUE. This indication is only made when Bit 0 of SetReferenceFilterOptionWord (page 267) is set to 1. In case Bit 0 of the SetReferenceFilterOptionWord is set to 0, then automatically ZX is also set to zero.

Note:

To be able to measure the length of the reference signal, the edge to be recognised is always changed internally. The time for the reprogramming is the debouncing time (Function SetReferenceHoldoffTime page 269). Signals which are shorter than the debouncing time or 1 millisecond cannot be measured and give a wrong value.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been read (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	The length of the reference signal.

Also refer to:

DisplayReferenceLengthFalse

Examples:

Declaration:

```
INST: DisplayReferenceLengthTrue;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

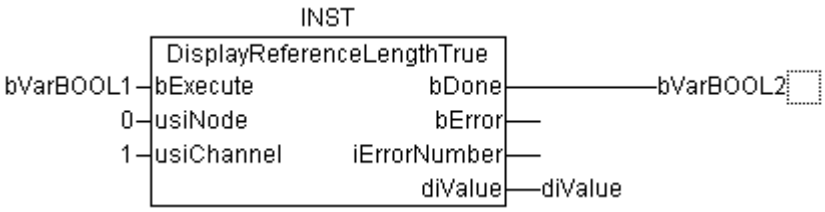
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1);
bVarBool2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



DisplaySnapshotPosition

PMCprimo-command: DS (**D**isplay **s**napshot position data)

Function library: **Display**

Description:

With this function block the last recorded SnapshotPosition before of the selected axis is given in increments (see DefinePositionSnapshot 249).

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been read (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	The stored PositionSnapshot.

Also refer to:

DefinePositionSnapshot

Examples:

Declaration:

```
INST: DisplaySnapshotPosition;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

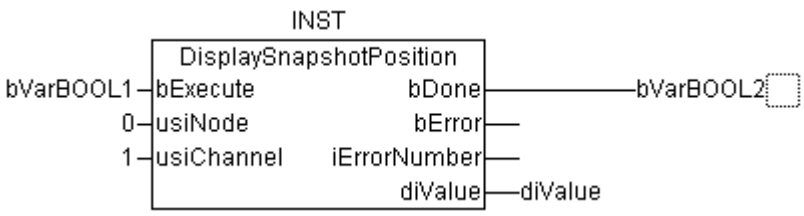
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1);
bVarBool2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



DisplayVelocity

PMCprimo-command: DV (**D**isplay **v**elocity)

Function library: **Display**

Description:

With this function block the actual speed of the selected axis is written to diValue. The unit of the speed is made in increments/second. When using the function block SetVelocityAveragingTime an averaged speed is determined via the set time.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been read (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	The actual speed

Also refer to:

SetVelocityAveragingTime

Examples

Declaration:

```
INST: DisplayVelocity;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

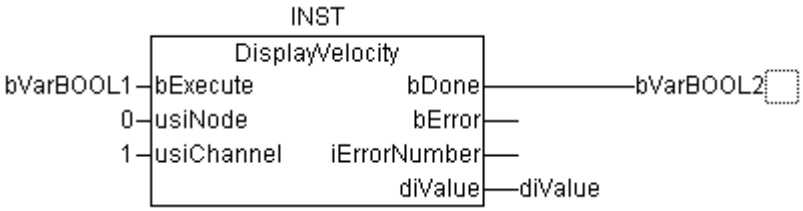
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1);
bVarBool2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



SetVelocityAveragingTime

PMCprimo-command: VT (Set **v**elocity averaging **t**ime constant)

Function library: **Display**

Description:

This function block is effective when using the function block DisplayVelocity and DefinePositionTriggerOutput. The set value specifies a period, over which PMCprimo determines the averaged speed of the selected axis. This averaged speed uses PMCprimo instead of the actual speed for the calculation of the phase displacement with electronic cams. For n=1 PMCprimo does not determine the average speed.

The determined average speed can be indicated with the function block DisplayVelocity, or displayed at the analogue output.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiValue (USINT)	The set period from 1 to 255 ms

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Also refer to:

DisplayVelocity and DefinePositionTriggerOutput

Factory setting: 1 millisecond

Examples:

Declaration:

```
INST: SetVelocityAveragingTime;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

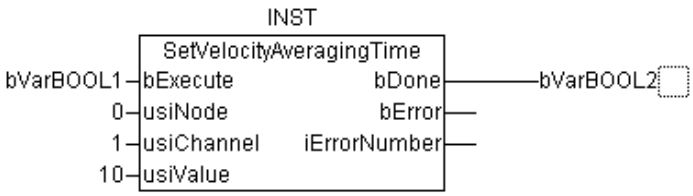
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiValue:=10)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1, usiValue:=10);
bVarBool2:=Inst.bDone;
```

Example in FBD:



14.9 Drive

DriveCommand

PMCprimo-command: “”

Function library: **Drive**

Description:

With the function block DriveCommand ASCII-Commands can be executed for the current control loop and speed controller of the drive. These commands are explained in a separate description. A possible application would be for example, to modify the speed control parameters during operation in order to compensate changing mass moment of inertias. However, the complete parameterisation of the controller can also be executed.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
sCommand (String)	The PMCtendo DD4 ASCII Command

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	A possible value from the servo controller (depending on the used command)

Examples:

Declaration:

```
INST: DriveCommand;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

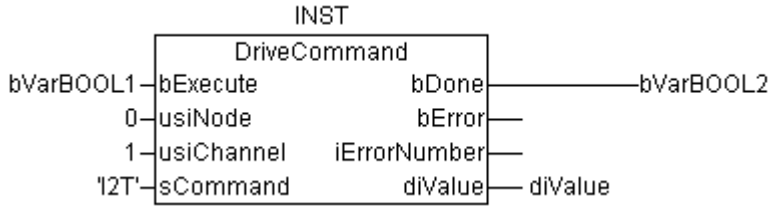
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, sCommand:='I2T')
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1, sCommand:='I2T');
bVarBool2:=Inst.bDone;
```

Example in FBD:



ASCIIToDrive

PMCprimo-command: QA

Function library: **Drive**

Description:

With the function block ASCIIToDrive you can open an ASCII channel to communicate with a PMCtendo DD4 drive. With the parameter usiCanNetwork the CAN-Net is selected. The parameter usiCanNode specifies the address of the PMCtendo DD4. It can communicate on the actual channel with the drive commands described before. The command is channel specific so you can communicate on every channel command with a different PMCtendo DD4.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiCanNetwork (USINT)	0: PMCtendo DD4 in CAN-Net, command must execute on the host node. 1: PMCtendo DD4 in CANOpen
usiCanNode (USINT)	The CAN node address (1 to 61)
sCommand (String)	The PMCtendo DD4 ASCII Command

Output variables:

BDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	A possible value from the servo controller (depending on the used command)

Examples:

Declaration:

```
ASCIIToDrive1: ASCIItoDrive;
bExecute2: BOOL;
bError2: BOOL;
iErrorNumber2: INT;
BDone2: BOOL
```

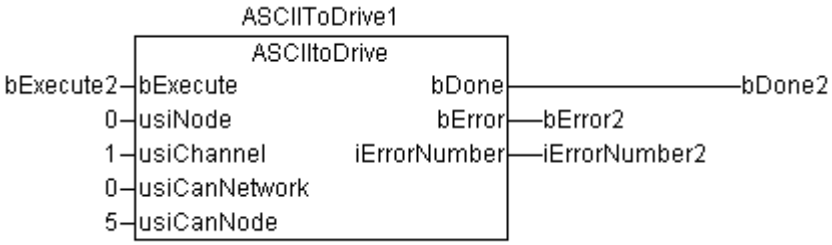
Example in IL:

```
CAL    ASCIIToDrive1(bExecute := bExecute2, usiNode := 0, usiChannel := 1, usiCanNetwork :=
0, usiCanNode := 5)
LD     ASCIIToDrive1.bDone
ST     bDone2
```

Example in ST:

```
ASCIIToDrive1(bExecute := bExecute2, usiNode := 0, usiChannel := 1, usiCanNetwork := 0,
usiCanNode := 5);
bDone2:= ASCIIToDrive1.bDone;
```

Example in FBD:



14.10 Encoder

DefineZeroMarkerInput

PMCprimo-command: DZ (**D**efine **z**ero marker input on/off)

Function library: **Encoder**

Description:

With this function block the zero marker of the encoder can be defined as reference signal. If 1 is indicated, the zero marker of the encoder is used by PMCprimo as a reference signal.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiDefine (USINT)	1 zero track is defined, 0 zero track is switched-off

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Factory setting: 0 (not activated)

Examples:

Declaration:

```
INST: DefineZeroMarkerInput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

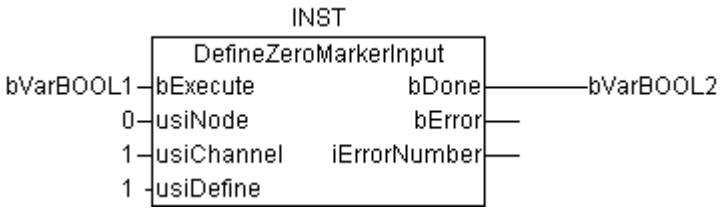
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiDefine := 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1, usiDefine := 1);
bVarBool2:=Inst.bDone;
```

Example in FBD:



SetEncoderFeedbackChannel

PMCprimo-command: FC (**F**eedback **c**hange encoder)

Function library: **Encoder**

Description:

With this function block an encoder input which is normally determined for another axis, can be deviated to the indicated axis. Shall for example an incremental encoder be connected with PMCprimo Drive on axis 1 is this only possible with the command FC:

An encoder is for example a possible application, which is fitted after a gearbox and shall be used as the actual value source for the motor. The encoder is connected at the free encoder input and is allocated to the axis 2 with PMCprimo Drive. Axis 1 (motor axis) can take over the positions of the axis 2 with this function and, therefore, the position control is calculated by this actual value.

The transport of goods which can slip is for example a further application. An additional encoder acquires the actual position of the product.

A reference input or the encoder zero marker must be defined at the axis the SetFeedbackChannel, SetFeedbackEncoder and NumberOfBits command refers to.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiValue (USINT)	0 no encoder information from another axis, 1 up to n (depending on the system) encoder information from the indicated axis

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Factory setting: 0 (switched off)

Examples:

Declaration:

```
INST: SetEncoderFeedbackChannel;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

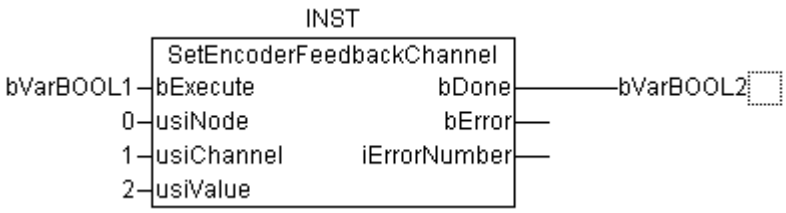
Example in IL:

```
CAL    INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiValue:= 2)
LD     INST.bDone
ST     bVarBOOL2
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1, usiValue:= 2);
bVarBool2:=Inst.bDone;
```

Example in FBD:



SetEncoderFilterTime

PMCprimo-command: PT (encoder filter)

Function library: **Encoder**

Description:

It is possible, to set the received encoder signals to an average via the set period. In case “unsmooth” encoder signals are received, these ones can get smooth herewith. This function block can only be used at an uncontrolled axis.

In case a torque peak is available with a master axis and with a certain position, this interference can also be seen at the slave axis. The influence can be removed or decreased by this filter.

It can only be reacted on the speed changes at the uncontrolled axis later by the information. Additionally they cause a static misalignment of the averaged position.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiValue (UINT)	0 no filter: 1 up to 10000 Milliseconds

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Also refer to:

SetEncoderScaling

Factory setting: 0 (switched off)

Examples:

Declaration:

```
INST: SetEncoderFilterTime;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

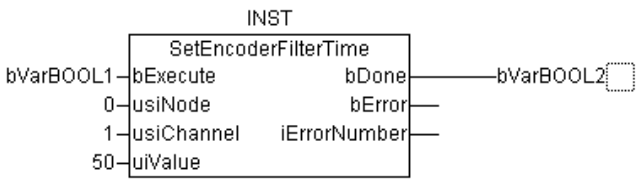
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue:= 50)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1, uiValue:= 50);
bVarBool2:=Inst.bDone;
```

Example in FBD:



SetEncoderScaling

PMCprimo-command: MS (encoder scaling)

Function library: **Encoder**

Description:

With this function block the determined position can be scaled. The position is multiplied by 2^{value}. This possibility achieves an increase of the resolution of the measuring system with the function block SetEncoderFilterTime. It is required, if a Slaveaxis with a high transmission ratio should follow this Masteraxis. In case of a ratio of 10:1 the Slave must run 10 increments with each increment of the Master. It is noticeable at the motor by noises and increased temperature development. In case the position is graduated and averaged the transmission ratio is decreased and this effect is reduced.

Typical values are 2 for the graduation and 8 for the information.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiValue (UINT)	Set graduation: multiplication with 2^{Value}, 0 no graduation, maximum 8

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Also refer to:

SetEncoderFilterTime

Factory setting: 0 (switched off)

Examples:

Declaration:

```
INST: SetEncoderScaling;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

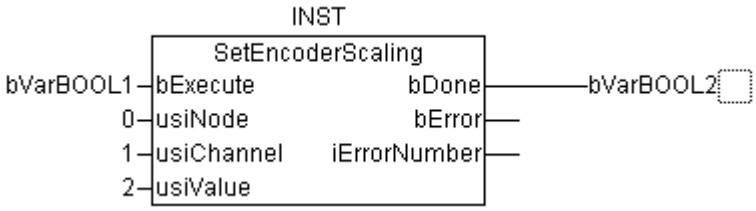
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiValue:= 2)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1, usiValue:= 2);
bVarBool2:=Inst.bDone;
```

Example in FBD:



SetEncoderTimeout

PMCprimo-Command: TO

Function library: **Encoder**

Description: (Set **t**ime**o**ut)

With this function block a timeout for the encoder signal is set. Depending on the pre-setting by Bit 6 of the function SetcontrolWord (page 182) the parameter is interpreted in a different way.

Bit 6 of the Function SetControlWord is 0:

After the start of a movement an encoder signal must be received by PMCprimo within the set monitoring time. If PMCprimo does not receive any encoder signal before the monitoring time has passed, PMCprimo switches off the controller release (motor is de-energised). The input of the monitoring time for encoder signals is made in the unit milliseconds.

Bit 6 of the Function SetControlWord is 1:

After the start of a movement an encoder signal must be received by PMCprimo within the set path (demand value specification). If PMCprimo does not receive any encoder signal within the set path, PMCprimo switches off the controller release (motor is de-energised). The input of the path for encoder signals is made in the unit increments.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiValue (UINT)	The monitoring time or way

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetcontrolWord

Factory setting: 500 Milliseconds

Examples:

Declaration:

```
INST: SetEncoderTimeout;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

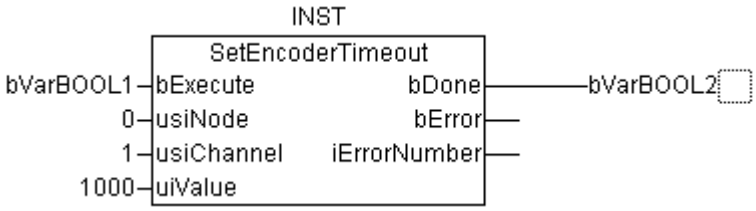
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue:= 1000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1, uiValue:= 1000);
bVarBool2:=Inst.bDone;
```

Example in FBD:



SetFeedbackEncoder

PMCprimo-command: FS (**F**eedback **s**et encoder type)

Function library: **Encoder**

Description:

With this function block the type of the encoder can be set for every axis. The following selectable possibilities for every axis are available.

- Incremental feedback encoder (SetFeedbackEncoder 0 - 2)
- SSI or Hiperface feedback encoder (SetFeedbackEncoder 5-8, 11-18)
- CANopen feedback encoder (SetFeedbackEncoder 9-10)
- Transmission of feedback encoder values with CANopen for PMCtendo DD4 (SetFeedbackEncoder 19-20)

With transmission of the demand positions with CANopen for PMCtendo DD4, the position control is done by PMCtendo DD4 (SetFeedbackEncoder 21-26).

SetFeedback Encoder	Option
0	Quadrature x 4
1	Quadrature x 2 (except channel 1 in PMCprimo Drive/2)
2	Quadrature x 1 (except channel 1 in PMCprimo Drive/2)
3	Reserved
4	Reserved
5	Hiperfache/SSI, relative position, binary; 300kHz
6	Hiperfache/SSI, relative position, gray code; 300kHz
7	Hiperfache/SSI, absolute position, binary; 300kHz
8	Hiperfache/SSI, absolute position, graycode; 300kHz
9	CANopen, relative position
10	CANopen, absolute position
11	SSI/Hiperface, relative position with high resolution (as of **version 1.009**)
12	SSI/Hiperface, absolute position with high resolution (as of **version 1.009**)
15	Hiperfache/SSI, relative position, binary; 100kHz
16	Hiperfache/SSI, relative position, gray code; 100kHz
17	Hiperfache/SSI, absolute position, binary; 100kHz

18	Hiperfache/SSI, absolute position, gray code; 100kHz
19	PMCtendo DD4, relative position (chose the PMCtendo DD4 with pd and consider the reference.)
20	PMCtendo DD4, absolute position (chose the PMCtendo DD4 with pd and consider the reference.)
21	CANopen, demand position, relative evaluation, positive reference signal (as of **version 2.000**) (chose the PMCtendo DD4 with pd consider the reference.)
22	CANopen, demand position, relative evaluation, negative reference signal (as of **version 2.000**) (chose the PMCtendo DD4 with pd and consider the reference.)
23	CANopen, demand position, absolute evaluation, offset set with command RF (page 271) (as of **version 2.000**) (chose the PMCtendo DD4 with pd and consider the reference.)
24	CANopen, demand position, relative evaluation, positive reference signal. The actual following error and the actual current is received. (as of **version 2.004**) (chose the PMCtendo DD4 with pd and consider the reference.)
25	CANopen, demand position, relative evaluation, negative reference signal The actual following error and the actual current is received. (as of **version 2.000**) (chose the PMCtendo DD4 with pd and consider the reference.)
26	CANopen, demand position, absolute evaluation, offset set with command RF (page 271). The actual following error and the actual current is received. (as of **version 2.000**) (chose the PMCtendo DD4 with pd and consider the reference.)

In case of modifications of for example 4096 to 2048 increments/revolutions the position control parameters change, too. In this case the factors will be double.

Enhancement as of **version 1.009**:

SetFeedbackEncoder 11 and 12, Hiperface relative and absolute:

In comparison with SetFeedbackEncoder 5,6 or 7,8 a higher resolution for the interchange of the demand values is used.

Enhancement as of **version 2.000**:

This extension operates only with **PMCtendo DD4 version 4.94** and higher.

SetFeedbackEncoder 21, 22 and 23 demand position with CANBus. The position control is made by PMCtendo DD4. The position control of PMCprimo (SetProportionalGainConstant, SetVelocityFeed-ForwardGain constant etc.) is not in use. With SetFeedbackEncoder 21 and 22 the latch function of PMCtendo DD4 can be activate. Therefore the drive command “IN2MODE” must set to 26 and reference input is input 2 of PMCtendo DD4. (SAVE+COLDSTART must be made)

The solution can set with the function block SetNumberOfBits command (page 83) (SetNumberOfBits 24 means 4096 increments per revolution. The rolling direction can be set with bit 5 of function block SetControlWord (page 182).

With SetFeedbackEncoder 21 and 22 the referencing is supported completely (InitialisePosition, SetContinuousReferenceMode, SetReferenceOptionsWord, SetReferenceFilterOptionsWord, SetReferenceFalseHighLimit, SetReferenceFalseLowLimit, SetReferenceTrueHighLimit, SetReferenceTrueLowLimit). There must set no DefineZeroMarkerInput or DefineReferenceInput. With FS21 or 22 no reference input of PMCprimo is allocated but referencing can be made.

Tip: With every PC command the actual position of the drive is read and the position counter of PMCprimo is set to it. If the motor is moved manhandled the change is seen after the next PC command. Therfore the position is not lost. After first switch on it isn’t made by relative position. The actual position in PMCprimo is still present. The behavior is the same like an incremental encoder.

As of **version 2.004**:

This extension operates only with PMCtendo DD4 **version 5.15** and higher.

New settings “SetFeedbackEncoder 24” to 26. As with “SetFeedbackEncoder 21” to 23 the demand position is sent to PMCtendo DD4 and the internal position loop of the PMCtendo DD4 is used. The referencing with “INMODE2 26” is also available. The difference is that the actual following error and the actual current is received.

To get the right value the drive commands “PGEARI” and “PGEARO “are changed automatically from PMCprimo to 1048576 (if “PRBASE” is 20) or 65536 (if “PRBASE” is 16). The setting of the following error (“PEMAX”) inside PMCtendo DD4 has to be adjusted by the customer.

The function should be only for service or adjusting. If the machine is finished then the “SetFeedbackEncoder” should be set to “SetFeedbackEncoder 21” to 23 to decrease the bus load

Overview FS settings and technical data:

FS	Description	Delay time for demand signal	Accuracy reference inputs	Number of synchronous CAN messages	PScope can display motor current	Following error available in PMCprimo	Use of position loop of PMCtendo DD4
0	Incremental encoder X4 multiplication PMCprimo Drive channel x.1: 4096 increments/turn	PMCprimo Drive Channel x.1: At the end of the 1ms Interrupt. Depending of processor load 200-300us Analogue output: In the 1ms interrupt after calculation of the position loop. Depending of processor load 200-300us	Hardware register < 1us	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
1	Incremental encoder X2 multiplication PMCprimo Drive channel x.1: 2048 increments/turn	see FS0	Hardware register < 1us	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
2	Incremental encoder X1 multiplication PMCprimo Drive channel x.1: 1024increments/turn	see FS0	Hardware register < 1us	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
5	SSI encoder : relative position, binary code PMCprimo Drive channel x.1 : Hiperface resolution with NB changeable relative position	see FS0	If PMCprimo Drive channel x.1 Hardware register < 1us With SSI encoder 1ms	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
6	SSI encoder : relative position, Gray code PMCprimo Drive channel x.1 : Hiperface resolution with NB changeable relative position	see FS0	If mcD channel x.1 Hardware register < 1us With SSI encoder 1ms	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
7	SSI encoder : absolute position, binary code PMCprimo Drive	see FS0	Referencing not possible	-	Yes, with PMCprimo Drive and channel x.1	Yes	No

FS	Description	Delay time for demand signal	Accuracy reference inputs	Number of synchronous CAN messages	PScope can display motor current	Following error available in PMCprimo	Use of position loop of PMCtendo DD4
	channel x.1 : Hiperface resolution with NB changeable absolute position						
8	SSI encoder : absolute position, Gray code PMCprimo Drive channel x.1 : Hiperface resolution with NB changeable absolute position	see FS0	Referencing not possible	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
9	CAN open encoder relative position	see FS0	Depending for CAN cycle time 1,2 or 4ms	1	Yes, with PMCprimo Drive and channel x.1	Yes	No
10	CAN open encoder absolute position	see FS0	Referencing not possible	1	Yes, with PMCprimo Drive and channel x.1	Yes	No
11	PMCprimo Drive channel x.1 resolution with NB changeable, resolution of demand speed with DPRam depends from NB setting, relative position	see FS0	Hardware register < 1us	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
12	PMCprimo Drive channel x.1 resolution with NB changeable, resolution of demand speed with DPRam depends from NB setting, absolute position	see FS0	Referencing not possible	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
15	see FS 5, frequency 100kHz	see FS0	If PMCprimo Drive channel x.1 Not available With SSI encoder 1ms	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
16	see FS 6, frequency 100kHz	see FS0	If PMCprimo Drive channel x.1	-	Yes, with PMCprimo	Yes	No

FS	Description	Delay time for demand signal	Accuracy reference inputs	Number of synchronous CAN messages	PScope can display motor current	Following error available in PMCprimo	Use of position loop of PMCtendo DD4
			Not available With SSI encoder 1ms		Drive and channel x.1		
17	see FS 7, frequency 100kHz	see FS0	Referencing not possible	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
18	see FS 8, frequency 100kHz	Depending for CAN cycle time. Transmission with the next synchronisation message. 1,2 or 4ms	Referencing not possible	-	Yes, with PMCprimo Drive and channel x.1	Yes	No
19	PMCtendo DD4 demand velocity and actual position. Relative position	see FS18	Depending for CAN cycle time 1,2 or 4ms	2	Yes, with version 3.56a or 4.94a	Yes	No
20	PMCtendo DD4 demand velocity and actual position. Absolute position	see FS18	Referencing not possible	2	Yes, with version 3.56a or 4.94a	Yes	No
21	PMCtendo DD4 demand position Relative position and positive reference signal	see FS18	Hardware register PMCtendo DD4 < 1us	1	No	No	Yes
22	PMCtendo DD4 demand position Relative position and negative reference signal	see FS18	Hardware register PMCtendo DD4 < 1us	1	No	No	Yes
23	PMCtendo DD4 demand position Absolute position	see FS18	Referencing not possible	1	Nein	No	Yes
24	See FS21 but with following error and actual current	see FS18	Hardware register PMCtendo DD4 < 1us	2	Yes, with version 5.x	Yes	Yes
25	See FS22 but with following error and actual current	see FS18	Hardware register PMCtendo DD4 < 1us	2	Yes, with version 5.x	Yes	Yes
26	See FS23 but with following error and actual current	see FS18	Referencing not possible	2	Yes, with version 5.x	Yes	Yes

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiValue (USINT)	The selected kind of the encoder

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetNumberOfBits

Factory setting: 0 (Quad incremental evaluation)

Examples:

Declaration:

```
INST: SetFeedbackEncoder;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

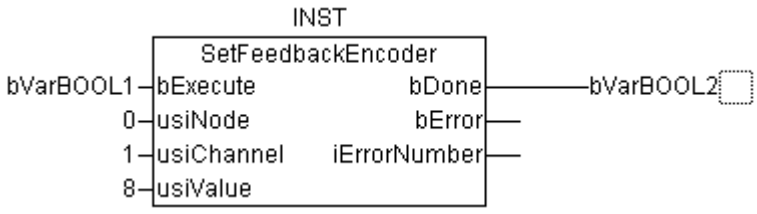
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiValue:= 8)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1, usiValue:= 8);
bVarBool2:=Inst.bDone;
```

Example in FBD:



SetNumberOfBits

PMCprimo-command: NB (Set **n**um**b**er of bits for SSI encoder)

Function library: **Encoder**

Description:

With this function block the number of the data bits can be set for every axis when using a SSI-, Hiperface®- or CAN-open-encoder. As a result SSI-, Hiperface®- and CAN-open-encoder types of simple 12bit Singleturn up to 25bit Multiturn models can be used..

Special feature with Hiperface®:

For Hiperface®-encoder always 12 Bit of the NB-value are reserved for the number of the revolutions. The remaining Bits indicate the resolution per revolution.
For NB28 for example the motor runs 65.536 increments per reveolution.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiValue (USINT)	The number of the entire Bits

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetFeedbackEncoder

Factory setting: 24

Examples:

Declaration:

```
INST: SetNumberOfBits;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

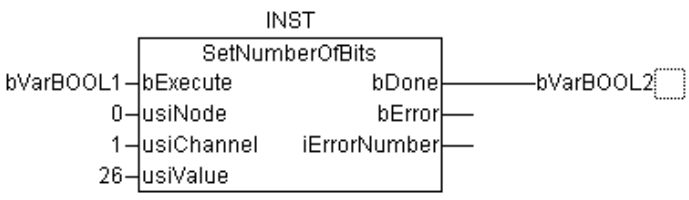
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiValue:= 26)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:= bVarBOOL1, usiNode:= 0, usiChannel:= 1, usiValue:= 26);
bVarBool2:=Inst.bDone;
```

Example in FBD:



14.11 Mapping

Generation and execution of tabular position allocations (=Map) or software-gear functions between axes in the Master-Slave-Operation.

An associated position of the slave axis is defined for every position of the master-axis by a tabular or linear position allocation.

The master axis can control an input, however, it is also possible that the master axis only works as receiver of encoder pulses (e.g. transmission shaft, vertical shaft) or as virtual axis (e.g. speed controlled indexing function).

If a position allocation is activated the corresponding position from the table is allocated to the slave axis as demand position.

It is sufficient in simple cases to define a linear allocation as software gear between master and slave axis. This software-gear function LINEAR is available in PMCprimo as tabular position allocation. The gear ratio master/slave can be set with the SM-command.

The tabular position allocation is used for more complex applications. It also counts, the more entries in the table, the more exact the master-slave allocation becomes, as PMCprimo carries out a linear interpolation between the various table positions.

Mechanical gear functions as curve gears or valve tappets on a camshaft can be simulated with the tabular position allocation and easily be replaced by the definition of a position allocation .

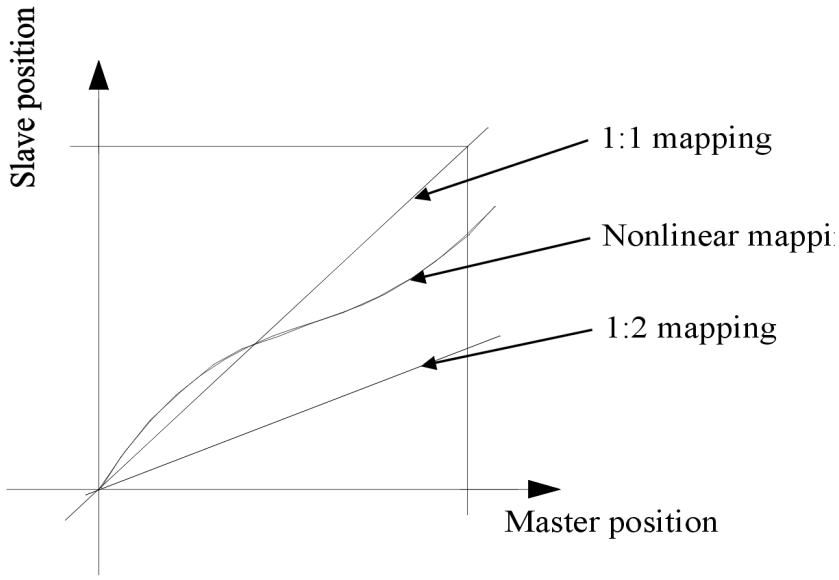


Illustration 10: Simple position allocations

If master and slave axes move within a given range (e.g. XY-table), a position allocation must be carried out for the required range. The position bound needs not to be set in this case (factory setting of SetBound can be maintained), as the motors do not continuously run in the same direction.

The following picture shows a typical tabular position allocation. The slave axis is for example a cutting tool and the master axis is for example the material transport. The speed of the master axis shall vary so that the material speed within the machine remains constant. The slave axis always runs to the corresponding demand position by the usage of a tabular position allocation independent on the current speed of the master axis (the cutting tool always remains in the correct position to the material transport).

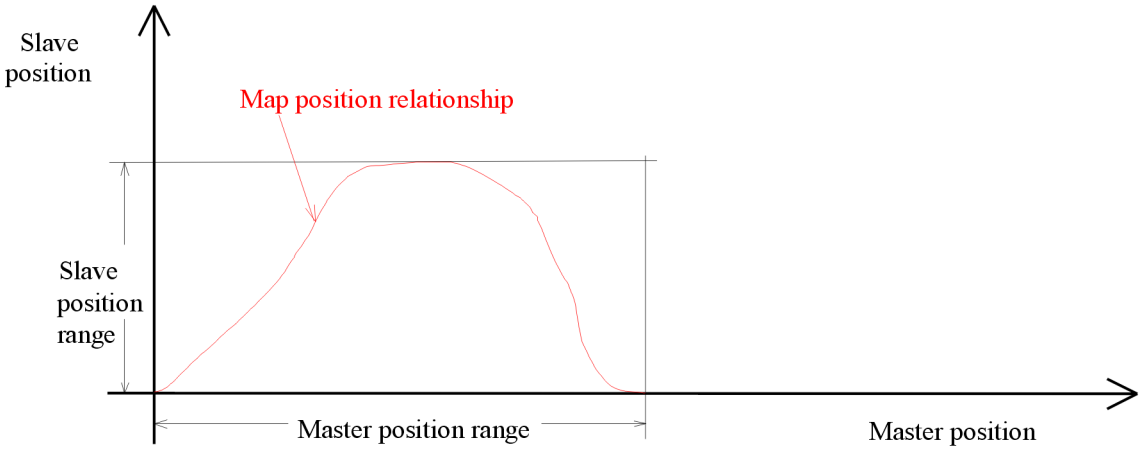


Illustration 11: Position allocation for a defined range

The position allocation which is shown in Illustration 11 can also refer to the application, in which the master axis behaves itself like a transmission shaft and the slave axis follows the shown position profile with every rotation of the transmission shaft (= vertical shaft). For this application the position bound of the master must be set to one rotation of the transmission shaft with the function SetPositionBound (page 133).

In case the master and/or slave axis behaves itself in a cyclic way the position bound must be set on every axis. The tabular position allocation must be defined in such a way that the position transitions at the cycle limits runs continuously. In case of intermittent transitions at the cycle limits sharp changes of the speed can occur at the slave axis.

In case the transition of the position values at the cycle limit of the slave axis is not continuous, the difference between the actual position bound and demand position bound effects a relative misalignment between slave and master axis. This error adds up over several cycles of the machine and behaves as constant drift of the slave axis. It is difficult to diagnose this problem, however, it can be avoided with a corresponding design of the tabular position allocation.

The tabular position allocation must cover the complete position bound of the master axis from 0 up to the cycle limit. The slave axis must know the position bound of the master axis enabling it to calculate its speed and its demand position beyond the cycle limit. It happens automatically when starting the execution of a tabular position allocation with the function MapLinkSlaveToMaster (page 104). The position bound of the master axis is automatically transmitted to the associated slave axis.

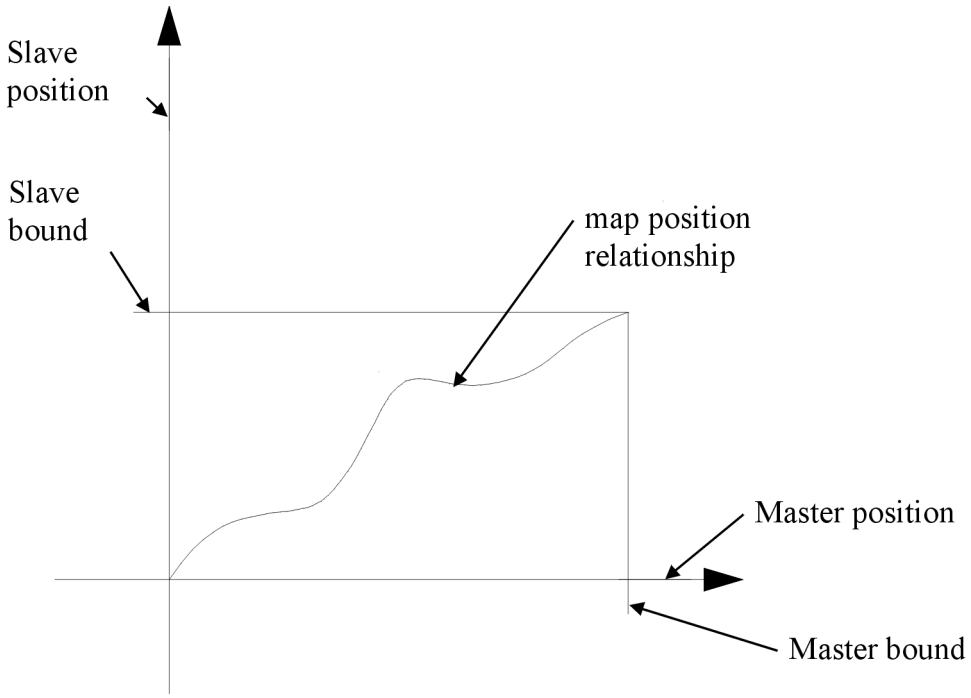


Illustration 12: position allocation for a cyclic machine

The diagram shown in Illustration 12 indicates a system in which the master and slave axes execute their cycles in the same time with identical position bounds, although they cover different distances. A coincidence of position bound and cycle time is not required. In practice neither position bound s nor cycle times coincide with linear software gear units, except of 1:1 gear units. The differences of position bound and cycle time between master and slave axes do not cause any problems when executing a tabular position allocation. It is also possible to exceed the position bound when executing a tabular position allocation with the slave position without problems. In this case the slave axis automatically compensates the zero point of your position bound when achieving the cycle limit.

The position allocation between master and slave axes can be loaded with an Offset-value with both axes. These Offset-values (SetMapBaseOffset and SetSlaveMapOffset (page 138) displace the relation master-slave along the position axis of the master or slave respectively. The value of the SetMapBaseOffset function is subtracted from the position of the master axis before the position enters the allocation. This effects a displace of the position allocation line to the right (see graph mentioned below). The value of the SetSlaveMapOffset function is added to the position of the slave axis, before the position enters the allocation effecting a displacement of the position allocation line upwards. The functions SetMapBaseOffset and SetSlaveMapOffset allow that one or all slave axes can be displaced or turned relatively to the master axis, even if the allocation master-slave is executed.

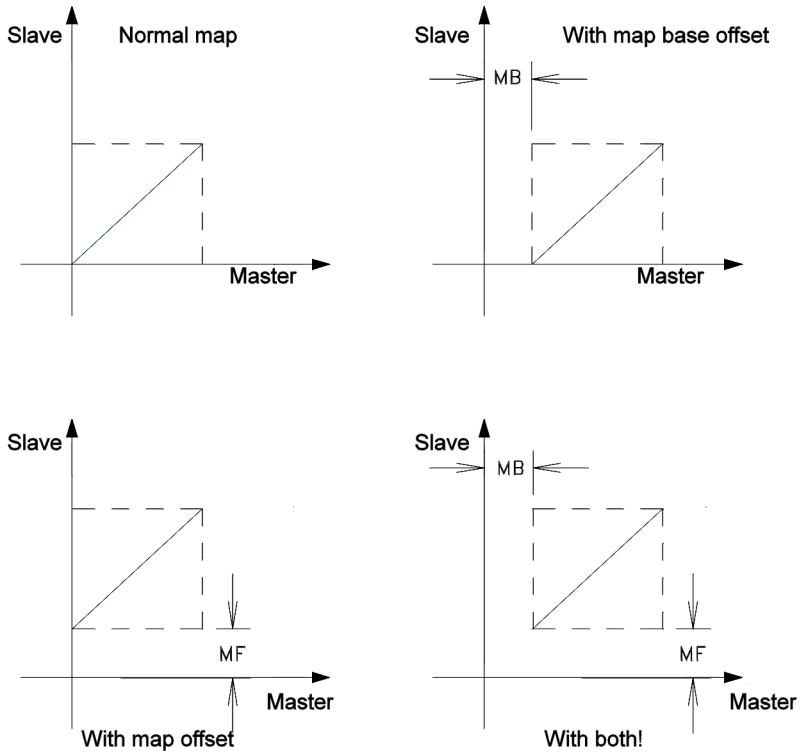


Illustration 13: effects positionoffset

The data of the position table are entered as absolute position of the slave axis. During the input of absolute positions for the slave axes each value represents the demand position of the slave axis in connection to the master axis. The associated master positions result from the position bound / number positions.

The table positions are centrally stored. Therefore, position allocations which are used in several axes must only be set once.

The data of the position allocation are transmitted to the particular axis with the function TransferMapData (page 139). This function must be uniquely executed before the first execution of the function ExecuteMap to the corresponding axis.

The first position of the master is always 0 during the enter of position allocations. Only the entered table values are stored. The values between the table positions are determined by linear interpolations, if the table step range is bigger than 1. This enables the user to prepare easily a position allocation with relatively few table values over a big allocation range without occupying much memory space.

Position allocations can internally be generated with the option “Motiongenerator”, if the option is released with the SK-command.

The software function differential allows the synchronisation of a drive in such a way that it can follow both the sum and the difference of the positions of two master drives.

This takes place analogue to a differential gear unit. The slave axis must be defined as slave of a master axis with the function MapLinkSlaveToMaster (page 104) and as slave of a second master axis (differential axis) with the function MapLinkSlaveToDifferentialMaster (page 102). The Bits 4 – 6 of the function SetMapLinkOptionsWord (page 201) determine, how the slave axis evaluates the positions of the master axis and Bit 0 of the function SetMapLinkOptionsWord determines, if the master or the differential axes transmit their demand or actual positions to the slave axis.

The status of the master axis is arbitrary as far as the definitions as slave axis are correctly executed. Additionally it is important that the settings with the LW-command (at the master and slave axes) have been carried out before executing the function MapLinkSlaveToMaster. Modifications at the LW-command are only effective after unlink and new link to the master channel.

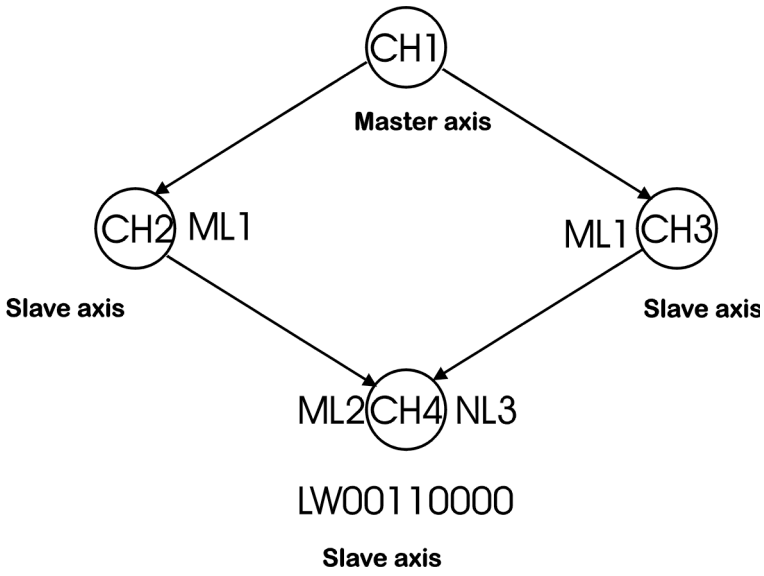


Illustration 14: Example for softwaredifferential (complex)

The different parameters for the execution of a position allocation can be summarised into the following equation. All parameters are to be set at the slave axis, except it is referred explicitly to the master axis.

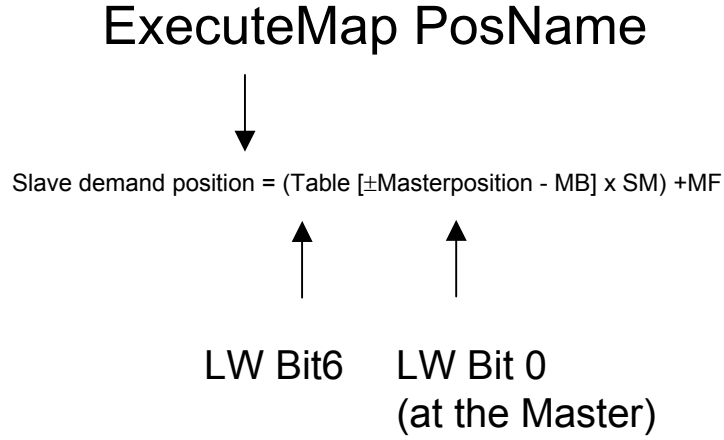


Illustration 15: Position allocation as equation

SM	SetScaleMapping
MB	SetMapBaseOffset
MF	SetSlaveMapOffset

Illustration 16: Position allocation with differential as equation

S	Demand position Slave axis
M_p	Position first master axis (MapLinkSlaveToMaster)
M_d	Position differential master axis (MapLinkSlaveToDifferentialMaster)
SB_{Master}	Position bound master axis
$SB_{Diffmaster}$	Position bound differential master axis

DefineMap

PMCprimo-command: EM (**E**dit **m**ap)

Function library: **Mapping**

Description:

With this function block a table allocation can be defined.

The program PMotion can be used for the generation of a prefabricated function block with all necessary variables. The definition is essentially facilitated with this program, as the curve generation is carried out graphically. From this data a function block can be exported, which only has one start input (bExecute) and internally carries out all necessary initialisations.

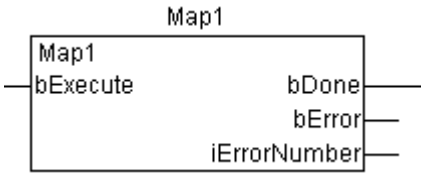
Input variables:

bExecute (BOOL)	The table allocation is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
sMapname (STRING)	The name of the table
diMasterBound (DINT)	The master position bound
iSteps (INT)	The number of the table support value (maximum 2000)
pdiSlavepos(POINTER)	Pointer to Array with the table support value

Output variables:

bDone (BOOL)	Table allocation (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see page 203)

Example of a function block from PMotion:



The function block (ST) itself is built-up in such a way:

Declaration:

```
FUNCTION_BLOCK Motion1
VAR_INPUT
        bExecute: BOOL;
END_VAR
VAR_OUTPUT
        bDone: BOOL:=FALSE;
        bError: BOOL;
        iErrorNumber: INT;
END_VAR
VAR
        Map1: DefineMap;
        iCount: INT;
        diSlavepos: ARRAY[1..201] OF DINT;
END_VAR
```

Implementation:

```
IF bExecute = TRUE AND bDone = FALSE THEN
        IF iCount = 0 THEN
                diSlavepos[1]:=0;
                diSlavepos[2]:=0;

                …
                diSlavepos[200]:=1000;
                diSlavepos[201]:=1000;
                Map1.bExecute:=TRUE;
                Map1.sMapName:='Motion1';
                Map1.diMasterBound:=1000;
                Map1.iSteps:=201;
                Map1.pdiSlavepos:=ADR(diSlavepos[1]);
                iCount:=1;
        END_IF
        Map1();
        bDone:=Map1.bDone;
        bError:=Map1.bError;
        iErrorNumber:=Map1.iErrorNumber;
END_IF
IF bExecute = FALSE  THEN
        Map1.bExecute:=FALSE;
        Map1();
        (* Reset FB *)
        bDone:=Map1.bDone;
        bError:=FALSE;
        iErrorNumber:=0;
        iCount:=0;
END_IF
```

ExecuteMap

PMCprimo-command: XM (E**x**ecute **m**ap)

Function library: **Mapping**

Description:

With this function block an available position allocation is activated. The activation is carried out at the slave axis. Before activation of the position allocation the master-slave relation must be defined with the function block MapLinkSlaveToMaster and the table must be loaded with the function block TransferMapData onto the axis. The position of the master axis is automatically, after connection of the slave axis with the master axis transferred to the same by the host. An active position allocation is signalled with “X” at the monitor with selected slave axis. The stop and abort commands de-activate a position allocation.

The linear position allocation is already pre-defined by the name **LINEAR**.

The position allocation LINEAR is especially used for the execution of transmission ratios specified with the function block SetScaleMapping.

Non-linear position allocations must be generated with the program PMotion as table or PMCprimo program and must be downloaded to the control.

The position allocation can be activated with stationary or moving master axes.

1. Activation with stationary master axis:

The slave axis moves according to the activated position allocation to the corresponding position which is allocated to the slave axis for the instantaneous position of the master axis. In case of an active software clutch (Bit 1 Function SetMapOptionsWord (age 101) the slave axis waits in the instantaneous position until the master axis is in the position corresponding to the position allocation and couples then. If the master runs a speed which is smaller/equal to 500Incr/sec the slave also couples without software clutch. The master speed should be brought to an average for that with the function SetMapBaseAdvanceTimeConstant (page 119).

2. Activation with moving master axis:

The activation of the position allocation with moving master axis must be executed by the aid of the software clutch (Bit 1 SetMapOptionsWord).

The execution of the function block ExecuteMap with activated web tension control with SetAnalogueControlMode is influenced by the Bits 4 and 5 of the function SetAnalogueControlWord (page 292).

For a speed allocation (Bit 4 command SetMapOptionsWord) only linear Maps (ExecuteMap LINEAR) are allowed.

Input variables:

bExecute (BOOL)	The position allocation is started in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
sMapname (STRING)	The name of the Map

Output variables:

bDone (BOOL)	Position allocation active (True) or coupling process or transient drive is not finished (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetMapOptionsWord, MapLinkSlaveToMaster, SetScaleMapping, SetMapBaseOffset and SetSlaveMapOffset

Examples

Declaration:

```
INST: ExecuteMap;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

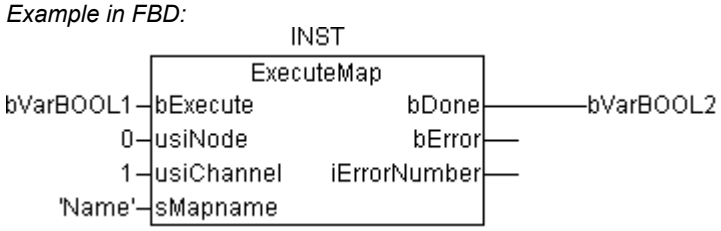
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, sMapname:= ‘Name’)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST (bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, sMapname:=’Name’);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



ExecuteMapVirtual

PMCprimo-Command: XV (e**x**ecute map **v**irtual)

Function library: **Mapping**

Description:

With this function block a slave position can be determined with the specified master position and the position table “Name”. As a result the slave position can be read for example in a variable without coupling. All settings (SetMapBaseOffset, SetSlaveMapOffset etc.) are observed analogue to ExecuteMap (page 94).

The position table must have been transmitted before with TransferMapData to the axis. MapLinkSlaveToMaster must have been executed at the axis.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue.
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
sMapname (STRING)	The name of the Map

Output variables:

bDone (BOOL)	Value has been read (True) or is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	The determined slave position

Connected function:

SetMapOptionsWord, MapLinkSlaveToMaster, SetScaleMapping, SetMapBaseOffset and SetSlaveMapOffset

Examples:

Declaration:

```
INST: ExecuteMapVirtual;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
diValue:DINT;
```

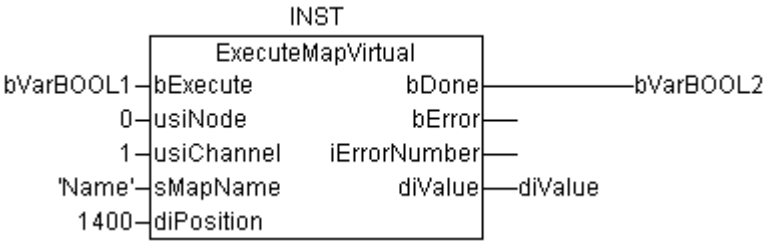
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, sMapname:= 'Name')
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, sMapname:='Name');
bVarBOOL2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



GetMappedMasterBound

PMCprimo-Command: GM (Get mapped master axis bound position)

Function library: **Mapping**

Description:

With this function block the position bound, which has been passed over from the master axis to the slave axis, can be read-out. This control function of the GM-command can be used to determine if the position allocation behaves as scheduled.

The calculated value can be displayed with the command GM. If mapping is not active, the SetPositionBound value (page 133) is displayed.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue.
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been read (True) or is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	The position bound of the master axis

Connected function:

SetMapOptionsWord, MapLinkSlaveToMaster, SetScaleMapping, SetMapBaseOffset and SetSlaveMapOffset

Examples:

Declaration:

```
INST: GetMappedMasterBound;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
diValue:DINT;
```

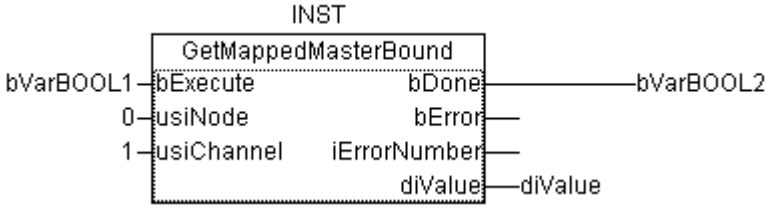
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1);
bVarBOOL2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



GetWrapAroundOffset

PMCprimo-Command: GW (Get wraparound offset value)

Function library: **Mapping**

Description:

With this function block the internally calculated offset value can be read-out at the slave axis during the wraparound (cycle limit). This control function can be used to determine if the position allocation behaves as scheduled. Additionally it can be determined if the behaviour of the slave axis during the wraparound is correct.

The value is internally calculated by the processor. The given value is only valid after activation of the position allocation. The reading of the GetWrapAroundOffset-value can only be carried out on the slave-axis.

The value must amount to the value 0 or +/- position bound in the simple case, when the position bound at the master and slave axes are identical. If the value changes upwards or downwards in course of time the axis drifts away. The settings of the position allocation must be checked in any case.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been read (True) or is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	The calculated Offset value during wraparound

Connected function:

SetMapOptionsWord, MapLinkSlaveToMaster, SetScaleMapping, SetMapBaseOffset and SetSlaveMapOffset

Examples

Declaration:

```
INST: GetWrapAroundOffset;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
diValue:DINT;
```

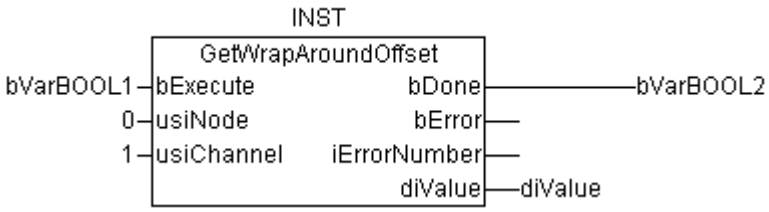
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1);
bVarBOOL2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



LengthOfAlignmentMove

PMCprimo-Command: XX (length of alignment move)

As of **version 2.004** available

Function library: **Mapping**

Description:

With this command the necessary alignment move for the position table “name” can be aquired, which are made automatically after the start of the function block ExecuteMap (Bit 0 of MapLinkSlaveToMaster [page 127] must be set to 0). Thus it is possible to check the alignment move before the execution of ExecuteMap. The function block must always run on the slave axis.

The position table must be tranferred to the axis with the command TransferMapData. On the axis MapLinkSlaveToMaster must be done.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue.
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
sMapname (STRING)	The name of the Map

Output variables:

bDone (BOOL)	Value has been read (True) or is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	The determined alignment move

Connected function:
SetMapOptionsWord, MapLinkSlaveToMaster, SetScaleMapping, SetMapBaseOffset und SetSlaveMapOffset

Examples:

Declaration:

```
INST: LengthOfAlignmentMove;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
diValue:DINT;
```

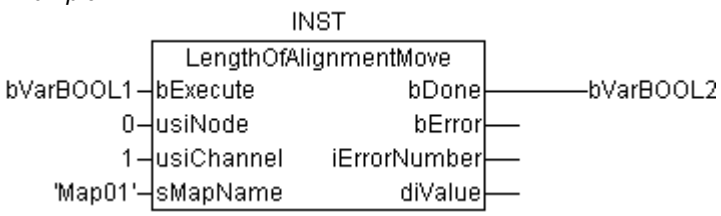
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, sMapname:= ‘Map01’)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.diValue
ST    diValue
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, sMapname:= Map01’);
bVarBOOL2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



MapLinkSlaveToDifferentialMaster

PMCprimo-Command: NL (Map link slave axis to differential master axis)

Function library: **Mapping**

Description:

With this function block the instantaneous axis is defined as the slave axis of a 2. master (differential). The value indicates the requested differential-master axis. This command must be used before activation of a position allocation if a differential master is necessary. Software differential also refer to the function on page 125.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiLinkNode (USINT)	The node number of the master axis
usiLinkChannel (USINT)	The axis number of the master axis

Output variables:

bDone (BOOL)	Definition closed (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetMapLinkOptionsWord

Factory setting: No master axis defined

Examples:

Declaration:

```
INST: MapLinkSlaveToDifferentialMaster;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

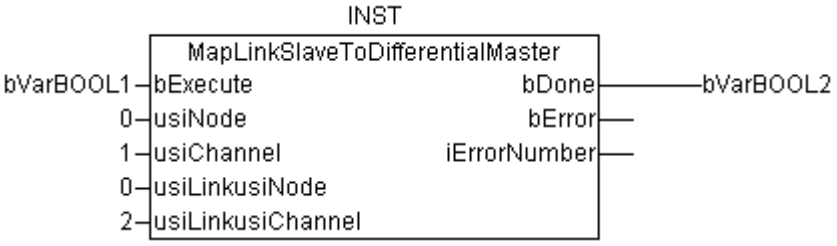
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiLinkNode := 0,
           usiLinkChannel := 2)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiLinkNode:=0, usiLinkChannel:=2);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



MapLinkSlaveToMaster

PMCprimo-Command: ML (**M**ap **l**ink slave axis to master axis)

Function library: **Mapping**

Description:

With this function block the instantaneous axis is determined as slave axis. The arguments indicate the requested master axis. This command must be used before activation of a position allocation in any case, as PMCprimo cannot execute a position allocation without definition of a master axis.

Input variables:

bExecute (BOOL)	The value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiLinkNode (USINT)	The node number of the master axis
usiLinkChannel (USINT)	The axis number of the master axis

Output variables:

bDone (BOOL)	Definition closed (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetMapLinkOptionsWord

Factory setting: no master axis defined

Examples:

Declaration:

```
INST: MapLinkSlaveToMaster;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

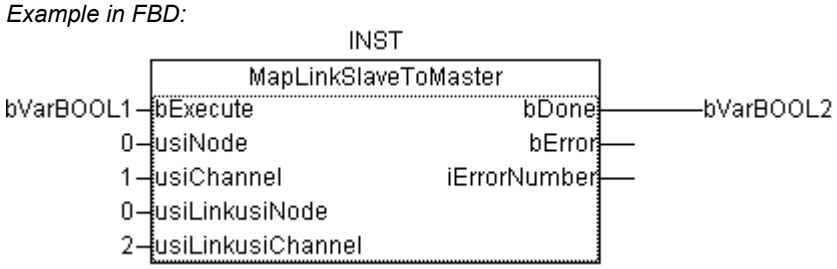
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiLinkNode := 0,
            usiLinkChannel := 2)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiLinkNode:=0, usiLinkChannel:=2);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



Motiongenerator

PMCprimo-Command: $M...

Function library: **Mapping**

Description:

With this function block a table allocation can be calculated by the internal motion generator.

The program PMotion can be used for generation of a pre-manufactured function block with all necessary variables. The definition is essentially facilitated by this program as the curve position is carried out graphically. A function block can be exported from these data which has one start input (bExecute) and internally carries out all required initialisation.

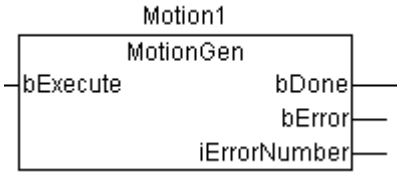
Input variables:

bExecute (BOOL)	The table is calculated in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 (corresponds to $MSTART *)
sMapName (STRING)	Name of the generated table (corresponds to $MNAME)
iSteps (INT)	Number of the supporting points (corresponds to $MNPT)
iNumberElements (USINT)	Number of the segments
pdiMasterpos (PDINT)	Pointer to the master positions (corresponds to $MMx)
pdiSlavepos (PDINT)	Pointer to the slave positions (corresponds to $MSx)
pdiFunction (PDINT)	Pointer to the relation functions (corresponds to $MFx)
pdiValuesA (PDINT)	Pointer to the marginal parameter (corresponds to $MAx)
pdiValuesB (PDINT)	Pointer to the marginal parameter (corresponds to $MBx)
pdiValuesC (PDINT)	Pointer to the marginal parameter (corresponds to $MCx)
pdiValuesW (PDINT)	Pointer to the marginal parameter (corresponds to $MWx)
pdiValuesX (PDINT)	Pointer to the marginal parameter (corresponds to $MXx)
pdiValuesY (PDINT)	Pointer to the marginal parameter (corresponds to $MYx)
pdiValuesZ (PDINT)	Pointer to the marginal parameter (corresponds to $MZx)

Output variables:

bDone (BOOL)	Table has been calculated (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Example of a function block from PMotion:



The function block (ST) itself is designed in such a way:

Declaration:

```
FUNCTION_BLOCK Motion2
VAR_INPUT
	bExecute: BOOL;
END_VAR
VAR_OUTPUT
	bDone: BOOL:=FALSE;
	bError: BOOL;
	iErrorNumber: INT;
END_VAR
VAR
	iCount: INT;
	Motion: Motiongenerator;
	sMapName: STRING:='Motion2'; (* $MName *)
	iSteps: INT:=1000; (* $MNPT *)
	iNumberElements: USINT:=1;
	diMasterpos: ARRAY [0..1] OF DINT := 0,1000; (* $MMx *)
	diSlavepos: ARRAY [0..1] OF DINT := 0,1000; (* $MSx *)
	diFunction: ARRAY [1..1] OF DINT := 9; (* $MFx *)
	diValuesA: ARRAY [1..1] OF DINT := 0; (* $MAx *)
	diValuesB: ARRAY [1..1] OF DINT := 0; (* $MBx *)
	diValuesC: ARRAY [1..1] OF DINT := 0; (* $MCx *)
	diValuesW: ARRAY [1..1] OF DINT := 12201110; (* $MWx *)
	diValuesX: ARRAY [1..1] OF DINT := 0; (* $MXx *)
	diValuesY: ARRAY [1..1] OF DINT := 0; (* $MYx *)
	diValuesZ: ARRAY [1..1] OF DINT := 0; (* $MZx *)
END_VAR
```

Implementation:

```
IF bExecute = TRUE AND bDone = FALSE THEN
	Motion.bExecute := TRUE;
	IF iCount = 0 THEN
		(* Set Variables *)
		Motion.sMapName := sMapName;
		Motion.iSteps := iSteps;
		Motion.iNumberElements := iNumberElements;
		Motion.pdiMasterpos := ADR(diMasterpos);
		Motion.pdiSlavepos := ADR(diSlavepos);
		Motion.pdiFunction := ADR(diFunction);
		Motion.pdiValuesA := ADR(diValuesA);
		Motion.pdiValuesB := ADR(diValuesB);
		Motion.pdiValuesC := ADR(diValuesC);
		Motion.pdiValuesW := ADR(diValuesW);
		Motion.pdiValuesX := ADR(diValuesX);
		Motion.pdiValuesY := ADR(diValuesY);
		Motion.pdiValuesZ := ADR(diValuesZ);
		iCount := 1;
	END_IF
	Motion( );
	bDone:= Motion.bDone;
	bError := Motion.bError;
	iErrorNumber := Motion.iErrorNumber;
END_IF
IF bExecute = FALSE  THEN
	iCount := 0;
	Motion.bExecute := FALSE;
	Motion( );
	(* Reset FB *)
```

```
        bDone := Motion.bDone;
        bError := FALSE;
        iErrorNumber := 0;
END_IF
```

This function block can be changed in such a way that for example single sections can be modified via variables. For that the function block generated by Pmotion must only be extended in a minimum way in CoDeSys. For that a new input variable is declared and this variable overwrites then the reserved value into the Arrays.

Extension for a further input variable:

Declaration:

```
FUNCTION_BLOCK Motion2
VAR_INPUT
        bExecute: BOOL;
        diEndposition; DINT;
END_VAR
… further Code not modified
```

Implementation:

```
IF bExecute = TRUE AND bDone = FALSE THEN
        Motion.bExecute := TRUE;
        IF iCount = 0 THEN
                (* Set Variables *)
                Motion.sMapName := sMapName;
                Motion.iSteps := iSteps;
                Motion.iNumberElements := iNumberElements;
                diSlavepos[1] := diEndposition;
                Motion.pdiMasterpos := ADR(diMasterpos);
                Motion.pdiSlavepos := ADR(diSlavepos);
                Motion.pdiFunction := ADR(diFunction);
                Motion.pdiValuesA := ADR(diValuesA);
                Motion.pdiValuesB := ADR(diValuesB);
                Motion.pdiValuesC := ADR(diValuesC);
                Motion.pdiValuesW := ADR(diValuesW);
                Motion.pdiValuesX := ADR(diValuesX);
                Motion.pdiValuesY := ADR(diValuesY);
                Motion.pdiValuesZ := ADR(diValuesZ);
        iCount := 1;
        END_IF
… further Code not modified
```

Of course it depends on the required modification, how many and which variables shall additionally be added and overwrite the specified values. The PMotion supplies only the basic structure in this case, in order to simplify the first programming.

SetAlignmentAcceleration

PMCprimo-Command: AA (Set map base/offset/scale factor **a**djustment **a**cceleration)

Function library: **Mapping**

Description:

The modification/introduction of a Positionoffset (SetMapBaseOffset and SetSlaveMapOffset page 123), or of the transmission ratio (SetScaleMapping page 109) results in a new soft position for the slave axis. PMCprimo can determine the acceleration ramp for achieving the alignment-velocity by the aid of the parameter. The indication of the parameter is made in increments / seconds^2. The parameter is to be set at the slave axis. If the parameter is 0, it is accelerated with a jump.

The equation velocity is set with the function SetMapAdjustmentVelocity.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diAcceleration (DINT)	The alignment acceleration

As of **Primo_V2_006.lib**: diAcceleration (UDINT)

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: An error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetMapBaseOffset, SetSlaveMapOffset and SetMapAdjustmentVelocity

Factory setting: 0 (jump to alignment velocity)

Examples:

Declaration:

```
INST: SetAlignmentAcceleration;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

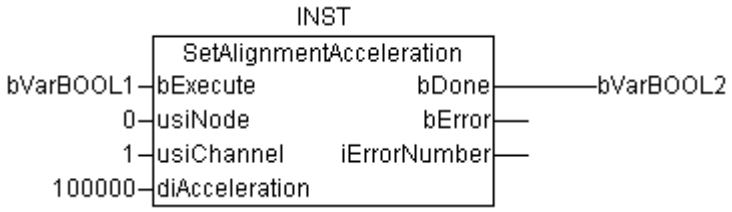
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, diAcceleration:=100000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, diAcceleration:=100000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetClutchLength

PMCprimo-Command: CL (Set clutch length)

Function library: **Mapping**

Description:

The software clutch length is only active, if Bit 5 of SetMapOptionsWord (page 127) is set to 1. If Bit 5 is set to 0 the software clutch time (SetClutchTime page 113) is active. This function block indicates the acceleration ramp as distance for the function of the software clutch. The function software clutch is activated with Bit 0, the function block SetMapOptionsWord. The software clutch function is designed for switching the slave axis to a driving master axis. The software clutch functions as a mechanical fixed point clutch which couples point-exactly while driving. The slave axis waits in the instantaneous position until the master axis achieves the driving position. For a velocity synchronous operation a value for n≠0 should be specified for the switching to the running master at the slave axis for SetClutchLength.

The value n=4194304 represents the longest possible and the value n=0 the shortest possible way as clutch length. The specification of the clutch length is executed at the slave axis, the indicated distance in increments, however, refers to the master axis.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diValue (DINT)	The coupling distance in increments of the master

Ab **Primo_V2_006.lib**: udiValue (UDINT)

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetMapOptionsWord and SetClutchTime

Factory setting: 1000 increments

Examples:

Declaration:

```
INST: SetClutchLength;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

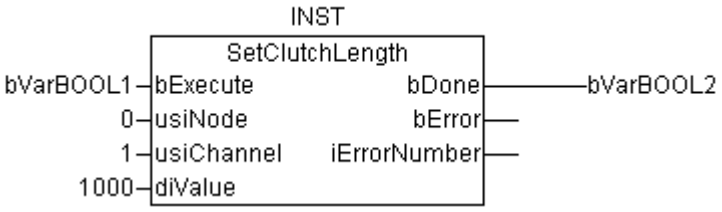
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, diValue:=1000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, diValue:=1000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetClutchTime

PMCprimo-Command: CT (Set **c**lutch **t**ime)

Function library: **Mapping**

Description:

The software clutch time is only active, if Bit 5 is set to 0 by SetMapOptionsWord (page 101). If Bit 5 is set to 1, the software clutch length (SetClutchTime page 113) is active. This function block indicates the acceleration ramp as time-slot pattern for the function of the software clutch. The function software clutch is activated with Bit 0 to the function block SetMapOptionsWord. The software clutch function is designed for connecting the slave axis to a driving master axis. The software clutch functions as a mechanical fixed point clutch which couples point-exactly while driving. The slave axis waits in the instantaneous position until the master axis achieves the driving position. For a velocity synchronous operation a value for n≠0 should be specified for connecting the running master at the slave axis for SetClutchTime.

The value n=20000 represents the longest and the value 0 the shortest clutch time. The clutch time is indicated in milliseconds.

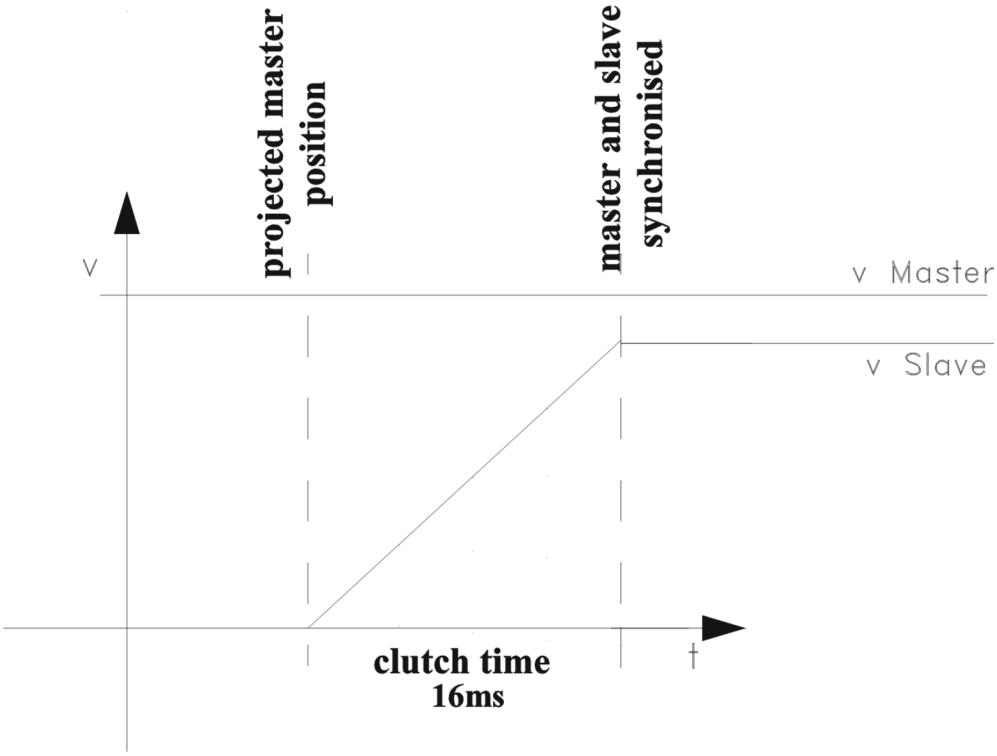


Illustration 17: Behaviour software clutch

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiClutchtime (UINT)	The coupling time in milliseconds Range of values: 1 to 20.000

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetMapOptionsWord and SetClutchLength

Factory setting: 1000 milliseconds

Examples:

Declaration:

```
INST: SetClutchTime;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

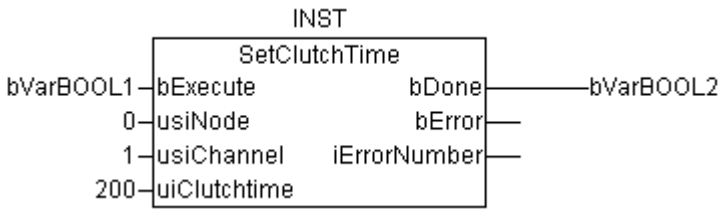
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiClutchtime:=200)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, uiClutchtime:=200);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetClutchWindow

PMCprimo-Command: CI (Set **c**lutch w**i**ndow)

Function library: **Mapping**

Description:

This command defines a window for software clutch.
If SetClutchWindow ≠ 0 and the way for alignment is lower than CI the slave makes an alignment move.
If SetClutchWindow ≠ 0 and the way for alignment is higher than CI the slave is using software clutch.
Bit 1 of the function block SetMapOptionsWord is ignored. But bit 2 and 3 is used for the clutch window. If the master is moving the clutch window is ignored. The clutch window can't used in speed mapping (bit4 of SetMapOptionsWord) and in tension control.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diValue (UINT)	position window

Ab **Primo_V2_006.lib**: uiValue (UINT)

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetMapOptionsWord

Factory setting: 0 increments

Examples:

Declaration:

```
INST: SetClutchTime;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

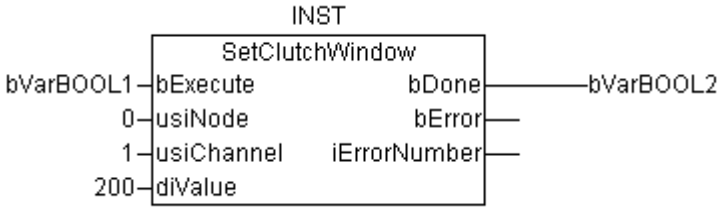
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiClutchtime:=200)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, uiClutchtime:=200);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetMapAdjustmentVelocity

PMCprimo-Command: AV (Set map base/offset/scale factor **a**djustment **v**elocity)

Function library: **Mapping**

Description:

The modification/introduction of the map base offset or map offset (SetMapBaseOffset and SetSlaveMapOffset page 138), or the scale map (SetScaleMapping page 136), provides a new demand position for the slave axis. PMCprimo can limit the maximum velocity during the alignment by the aid of the factor. A position modification is not aligned with a single non-sequential demand value change in case of a modification/introduction of positionoffset/transmission ratio with a factor $\neq 0$. The alignment velocity is indicated in percentage starting from the actual demand velocity. If the actual demand velocity is 0, the alignment is executed with the velocity on the basis of the function SetSlowSpeed

The modification of the scale map ratio with an active position allocation at the slave axis also requires a velocity modification. The function acts in this case exactly as with the modification of a position offset. The value is to be given at the slave axis.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiPercent (USINT)	The velocity from 1 to 200 percent , 0 jump

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetAlignmentAcceleration, SetSlowSpeed, SetScaleMapping, SetMapBaseOffset and SetSlaveMapOffset

Factory setting: 0 (jump to a new position)

Examples

Declaration:

```
INST: SetClutchTime;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

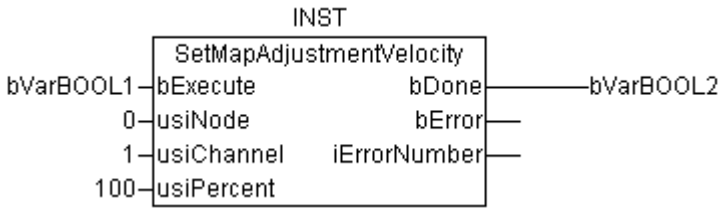
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiPercent:=100)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiPercent:=100);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetMapBaseAdvance

PMCprimo-Command: BA (Set map base advance)

Function library: **Mapping**

Description:

With this function block an offset of the master position, set with a SetMapBaseOffset (see page 123) can be provided with a velocity dependent phase shifting.

$$\text{map base advance} = \frac{\text{master speed}}{256} \times \frac{BA}{256}$$

The velocity of the master axis is indicated in increments/second, for example a phase shifting of 30 increments is the result with a velocity of 10.000 increments / second and a SetMapBaseAdvance-value of 200.

The velocity of the master axis can be averaged temporally. For that the function and SetMapBaseAdvanceTimeConstant exists with which the time for determination average velocity can be set.

This factor can also be considered as set time. A value of 1 corresponds to 15,625 microseconds. A phase shifting of 64 corresponds to 1 milliseconds and, therefore, the distance, the master makes within this time.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiPhase (UINT)	The velocity dependent phase shifting from –65535 to 65535

As of **Primo_V2_006.lib**: diPhase (DINT)

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetMapBaseOffset and SetMapBaseAdvanceTimeConstant

Factory setting: 0 (switched-off)

Examples:

Declaration:

```
INST: SetMapBaseAdvance;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

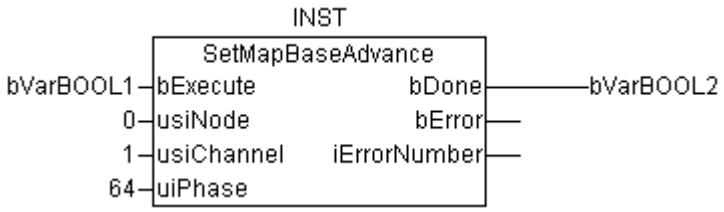
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiPhase:=64)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, uiPhase:=64);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetMapBaseAdvanceTimeConstant

PMCprimo-Command: BT (Set base advance time constant)

Function library: **Mapping**

Description:

This function block is effective when using the module SetMapBaseAdvance. The value specifies a period, during which PMCprimo determines the averaged velocity of the axis. This average velocity uses PMCprimo instead of the actual velocity for calculating the phase shifting of a Offset of the master position at the slave axis. PMCprimo does not determine any averaged velocity of diValue=1.

The velocity brought to an average is also used in order to notice if the master drives with the function ExecuteMap (see page 92).

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiValue (UINT)	The time from 1 to 10.000 milliseconds

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetMapBaseOffset and SetMapBaseAdvanceTimeConstant

Factory setting: 1 millisecond

Examples:

Declaration:

```
INST: SetMapBaseAdvanceTimeConstant;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

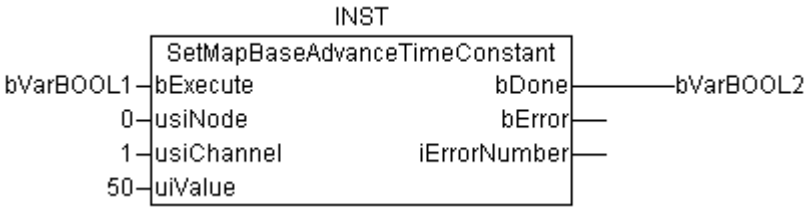
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue:=50)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, uiValue:=50);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetMapBaseOffset

PMCprimo-Command: MB (Set **m**ap **b**ase offset for master map positions)

Function library: **Mapping**

Description:

This function block allows the introduction of an offset to influence the position allocation between master and slave axis. The value is subtracted from the position of the master axis (it is the resulting position when using a softwaredifferential). This effects a shifting of the position allocation line. The position allocation line normally runs between the zero and final points. If the value changes during a position allocation, an alignment is executed with the velocity defined over the SetMapAdjustmentVelocity and the alignment acceleration SetAlignmentAcceleration (page 109).

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diValue (DINT)	Offset to be adjusted

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetMapBaseOffset and SetMapBaseAdvanceTimeConstant

Factory setting: 0 (switched off)

Examples:

Declaration:

```
INST: SetMapBaseOffset;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

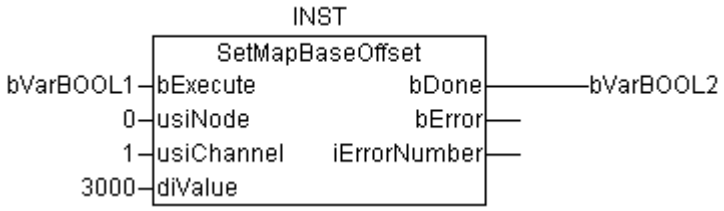
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, diValue:=3000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, diValue:=3000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetMapLinkOptionsWord

PMCprimo-Command: LW (Set map link options word)

Function library: **Mapping**

Description:

With this function block the axle-coupling between the slave and master axes is adjusted. The adjustment is executed on the slave axis and must be made before definition as slave axis with the function MapLinkSlaveToMaster (page 104).

With the pre-adjustment axle-coupling the function Softwaredifferential can be released and determined. With an active function the demand position is achieved from the sum or difference of two master axes. The 2. master axis must be defined with the function block MapLinkSlaveToDifferentialMaster (page 102). The function can released and determined with the Bits 4 to 6.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiControlword (USINT)	Each Bit has the following meaning:

Bit 0: This Bit determines, if the slave axis shall follow the demand or actual values of the master axis.

0: The slave axis follows the demand positions of the master axis.

1: The slave axis follows the actual positions of the master axis.

Bit 2: New function as of **version 2.000**
In speed mapping it is possible to restrict the slave velocity to a maximum (value of SetVelocity) and a minimum (value of SetSlowSpeed). The bits 2 and 3 of SetMapOptionsWord for definition of direction are active in this case.

0: Function speed limit is not active.

1: Function speed limit is active, if bit 4 of MW is set to 1

Bit 2: not occupied.

Bit 3: not occupied.

Bit 4: This Bit releases the function Softwaredifferential. The sum or the difference of two independent master positions determine the demand position of the slave axis.

0: Function Softwaredifferential closed

1: Function Softwaredifferential released.

Bit 5: This Bit determines, if the demand position of the slave axis for the function Softwaredifferential is the sum or difference of the Masterpositions.

0: The Masterposition are added.

1: The position of the 2. Master axis is subtracted from the 1. Master axis.

Bit 6: This Bit determines if the sign for the demand position of the slave axis is negated for the function Softwaredifferential.

0: The polarity sign is not negated.

1: The polarity sign is negated.

Bit 7: not occupied.

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

MapLinkSlaveToMaster and MapLinkSlaveToDifferentialMaster

Factory setting: 0

Examples:

Declaration:

```
INST: SetMapLinkOptionsWord;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

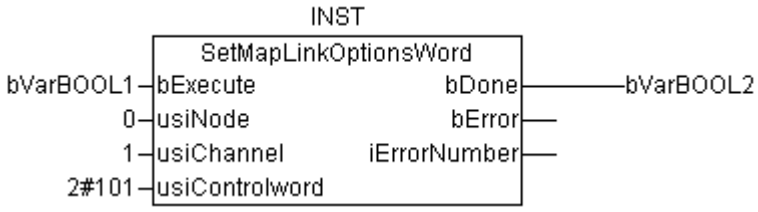
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiControlword:=2#101)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiControlword:=2#101);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetMapOptionsWord

PMCprimo-Command: MW (Set **m**ap options **w**ord)

Function library: **Mapping**

Description:

With this function block the behaviour of the position allocations is preset at the slave axis:

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiControlword (USINT)	Each Bit has the following meaning:

Bit 0: This Bit determines the behaviour of the slave axis when activating a position allocation.

0: Software clutch not active, i.e. the slave axis drives to the demand position assigned to it when activating a position allocation.

1: Software clutch active, i.e. the slave axis waits until the master axis takes the associated position, and afterwards accelerates to the required velocity within the set time (SetClutchTime) in order to comply the position allocation.

Enhancement as of **version 2.002**:

With speed mapping (bit 4 of command "**SetMapOptionsWord**" must set) it is possible now to go in mapping without software clutch. Therfore the velocity of master must be lower than 500 increments per second.

Bit 1: This Bit determines if a necessary compensating movement (for example after introduction of SetMapBaseOffset or SetSlaveMapOffset) is independent or dependent on the set position bound. Example: The axis position is –10000 increments and the position bound is 10000 increments. In case of an input of SetMapBaseOffset+20000 there is a compensating movement of 0 increments, when Bit 1 is set.

0: The position bound does not have any influence to a compensating movement.

1: The position bound is considered when executing a compensating movement. Always the shorter compensating drive is selected.

Bit 2: This Bit determines, if the demand position is only driven in the direction selected by Bit 3 when activating a position allocation.

0: Direction for starting the demand position not defined.

1: Direction for starting the demand position can be defined with Bit 3.

Bit 3: This Bit determines the starting direction of the demand position when activating a position allocation, in case Bit 2 is set to 1.

0: The correction movement is executed in a positive direction.

1: The correction movement is executed in a negative direction.

Bit 4: Selection between position allocation and velocity allocation. A velocity allocation is meaningful, when the absolute position allocation is without meaning.

0: Position allocation

1: Velocity allocation

Enhancement as of **version 2.002**:

The master velocity can be averaged in speed mapping with the command "**SetMapBaseAdvanceTimeConstant**". Damping of rapid movements of the master is possible with it.

Enhancement as of **version 2.005**:

Now it is possible to interrupt speed mapping with stop to position. In this case bit 6 of command SetMapOptionsWord is ignored.

Bit 5: Coupling can be executed via a set time or via a set clutch length.

0: Coupling is executed with the set clutch time SetClutchTime (page 113)

1: Coupling is executed with the set clutch length SetClutchLength (page 111).#

Enhancement as of **version 2.004**:

When "**StopMotor**" is used the map is still active when decelerating. The setting of "**SetClutchLength**" is always used. This means the bit 5 of "**SetMapOptionsWord**" (SetClutchTime / SetClutchLength setting) is ignored because with "**SetClutchTime**" it is not possible to reach the target position.

Bit 6: It is possible, to uncouple with MoveToAbsolutePosition or MoveRelativePosition. The velocity can be selected for that.

0: It is driven with the actual slave velocity.

1: It is driven with the velocity set by SetVelocity (page 240).

Enhancement as of **version 2.004**:

Clutching out with StopMotor the map is done until stop if the bit is set. Only the clutch length SetClutchLength is used, bit 5 of command SetMapOptionsWord has no relevance.

Bit 7: As of **version 1.006a:**
Activate an automatic bound correction. The slave bound is set automatic to the slave position at the master bound and its scale map. The actual bound can be displayed with function GetMappedMasterBound (page 96).
Enhancement as of **version 2.000**
If the axis is not moving in mapping an automatic bound correction is possible with this command. Therefore the bound set by SB is multiplied with the scale map (SetScaleMapping). With an odd gear transmission a reference sensor is no longer necessary.
Example: SB4096; SM1,3
The bound for it is 1365,33. Therefore a drift of one increment every 3 bound would happened. With correction the bound set two cycles to 1365 and one cycle to 1366

Special case:

In case Bit 0 and Bit 2 are set simultaneously, this defines the following behaviour:

The slave couples immediately. As a result a misalignment arises which is stored as Offset for the slave. The misalignment can be compensated by SetSlaveMapOffset 0 later on.

Enhancement as of **version 2.006**:
If position mapping is set then the offset through the change of map ratio (SetScaleMapping) is stored as Offset for the slave.

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError)

Connected functions:

MapLinkSlaveToMaster and MapLinkSlaveToDifferentialMaster

Factory setting: 0

Examples:

Declaration:

```
INST: SetMapOptionsWord;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

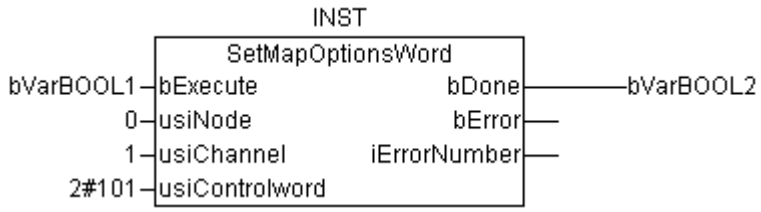
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiControlword:=2#101)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiControlword:=2#101);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetMapPositionTimeout

PMCprimo-Command: MT (Masterposition timeout)

Function library: **Mapping**

Description:

If master values are transmitted in a linked system, it is executed in a cycle of 4 milliseconds. This position is interpolated for the cycle time of one milliseconds. For monitoring the transfer of the master position via the CANopen-network a monitoring time can be adjusted by this function block. If the master values are not received within the adjusted time an error message is signalled and the motor is switched-off.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiValue (UINT)	The time from 4 up to 10.000 milliseconds As of **version 2.006** new range of values: 0 to 1000

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Factory setting: 4 milliseconds

Examples:

Declaration:

```
INST: SetMapPositionTimeout;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

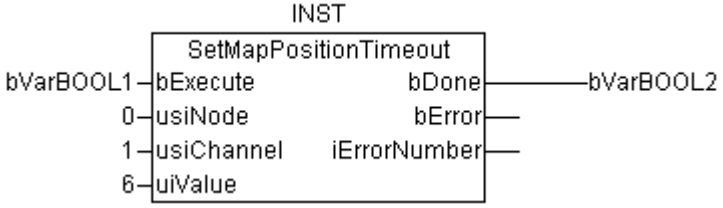
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue:=6)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, uiValue:=6);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetMapScaleFromBounds

PMCprimo-Command: BR (Set map scale factor from **b**ounds **r**atio)

Function library: **Mapping**

Description:

With this function block the transmission ratio (SetScaleMapping page 136) can automatically be calculated from the specified position bounds of the master and slave axes. The calculated transmission ratio is equal to the value SetScaleMapping (slave)/SetScaleMapping (master).

Input variables:

bExecute (BOOL)	The transmission ratio is calculated in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetScaleMapping

Examples:

Declaration:

```
INST: SetMapScaleFromBounds;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

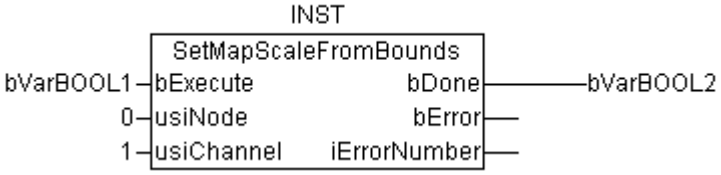
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetPositionBound

PMCprimo-Command: SB (**S**et position overflow **b**ound)

Function library: **Mapping**

Description:

This function block defines a position bound as limitation of the absolute position, i.e. if the motor position exceeds the adjusted position bound, the adjusted value is subtracted from the absolute value of the position counter.

The limitation of the absolute position is mostly identically adjusted with the reference position. This facilitates the continuous monitoring of the reference signals and the execution of the necessary corrections resulting from them.

Each complete position bound (limit value exceeding) is counted in an overflow counter. This counter can be cancelled or set with the function SetPositionOverflowCounter (page 135).

A typical application for the usage of position bounds is a cyclic or rotating movement where only the position within one rotation or cycle is important. Position indications outside of the selected position bound with travelling commands are possible at any time, however, the actual position can never be bigger than the position bound (subtraction of the adjusted value from the position counter). The overflow counter includes the number of the completely driven cycles.

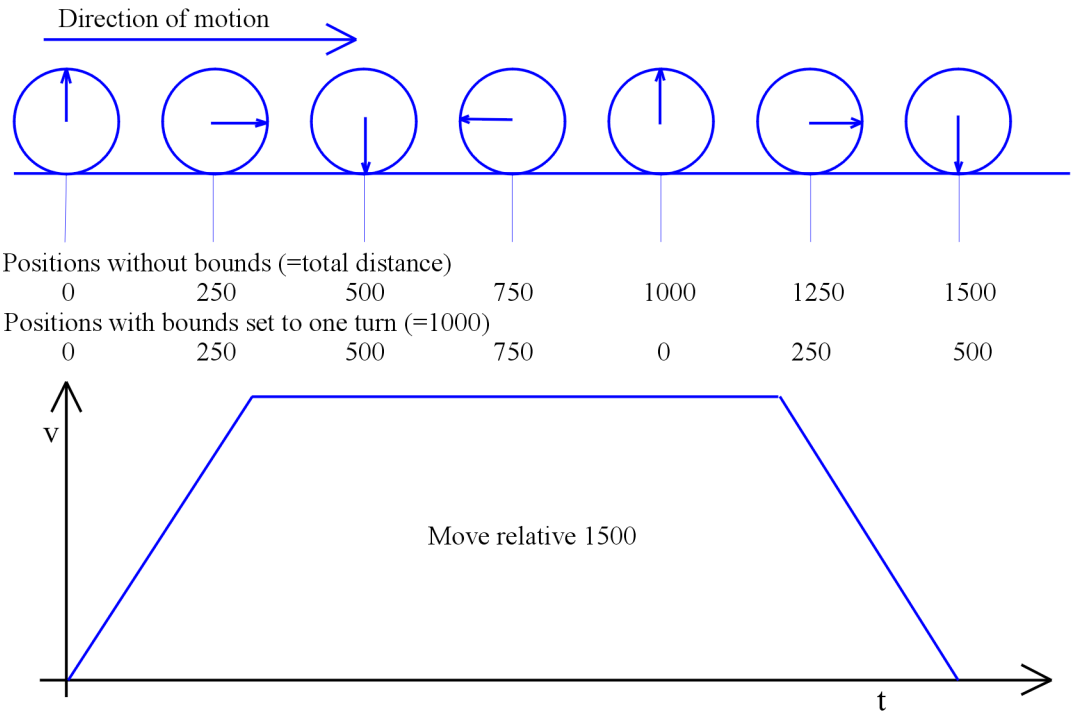


Illustration 18: position bound

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
udiValue (UDINT)	The position bound from 1 to 4.000.000 increments As of version **2.004** new range : 1 to 2.000.000.000

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetPositionOverflowCounter

Factory setting: 4.000.000 increments
As of version **2.005** new *Factory setting:* 4.194.304
This helps to avoid problems with absolute encoders because therefore the bound must have a value of 2^n.

Examples:

Declaration:

```
INST: SetPositionBound;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

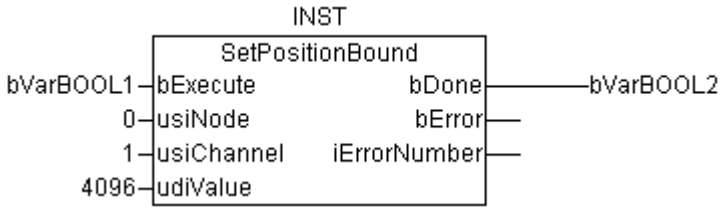
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, udiValue := 4096)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, udiValue := 4096);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetPositionOverflowCounter

PMCprimo-Command: BC (Set position overflow **c**ounter)

Function library: **Mapping**

Description:

This function block locates the overflow counter to the specified value. The overflow counter counts the exceeding of the position bounds. The counting value is incremented in case of exceeding in positive and decremented when exceeding the cycle limitation in negative direction.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diCounter (DINT)	The count of a counter

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:
SetPositionCounter

Factory setting: 0

Examples:

Declaration:

```
INST: SetPositionOverflowCounter;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

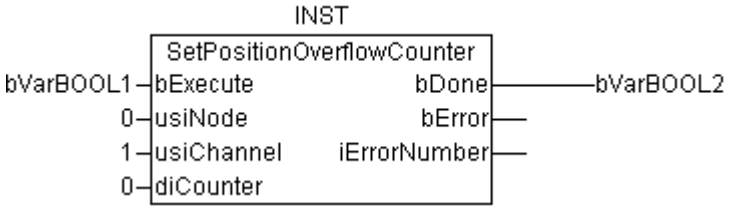
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, diCounter := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, diCounter := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetScaleMapping

PMCprimo-Command: SM (**S**cale **m**apping)

Function library: **Mapping**

Description:

With this function block the transmission ratio can be adjusted between the master and slave axes. The adjustment of the transmission ratio is executed at the slave axis. The absolute Slave-Position is multiplied with the counter and divided with the denominator. In case the transmission ratio changes with an active position allocation, the velocity for the equation with the parameter SetMapAdjustmentVelocity (page 117) and the acceleration with the value of the parameter SetAlignmentAcceleration (page 109) is determined in case of the change-over of the position at the slave axis.

Example: transmission ratio 1096/361

⇒ The actual demand position of the slave axis is the demand position from the position allocation multiplied with the factor 1096/361.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiCounter (UINT)	The counter 0 up to 65535 As of version **2.005** new *Factory setting:* 400.000
uiDivider (UINT)	The denominator 1 up to 65535 As of version **2.005** new *Factory setting:* 400.000

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetMapAdjustmentVelocity and SetAlignmentAcceleration

Factory setting: counter 1, denominator 1

Examples:

Declaration:

```
INST: SetScaleMapping;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

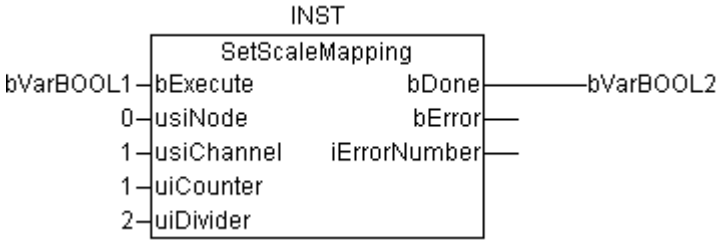
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode :=0, usiChannel :=1, uiCounter :=1, uiDivider :=2 )
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, diCounter := 0, uiDivider :=2);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetSlaveMapOffset

PMCprimo-Command: MF (Set slave **m**ap position o**f**fset)

Function library: **Mapping**

Description:

This function block allows the introduction of an offset for influencing the position between master and slave axes. The value is added to the demand position of the slave axis allowing a shift of the position allocation line relative to the position of the master axis. The position allocation line normally runs between the zero and final points. If the value changes during a position allocation an alignment with the velocity defined via the function SetMapAdjustmentVelocity and the alignment acceleration SetAlignmentAcceleration (page 109) is executed.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diValue (DINT)	The Offset to be adjusted +/- 4.000.000

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetMapAdjustmentVelocity and SetAlignmentAcceleration

Factory setting: 0 (switched off)

Examples:

Declaration:

```
INST: SetSlaveMapOffset;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

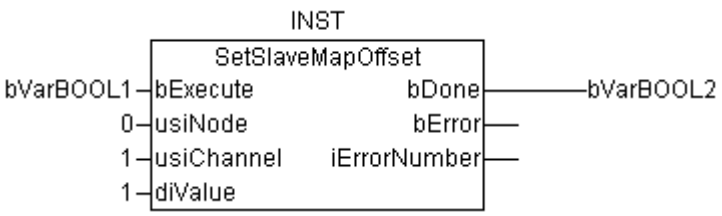
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, diValue := 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, diValue := 1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



TransferMapData

PMCprimo-Command: TM (**T**ransfer **m**ap data)

Function library: **Mapping**

Description:

With this function block a prepared position table is transferred from the Host to the selected axis.

The position table itself is transferred in the flash of the Host and must be transferred to the axis again if required. This is the case for example, if it is calculated from the Mapgenerator a new.

Input variables:

bExecute (BOOL)	The position table is transmitted in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
sMapname (STRING)	The name of the Map

Output variables:

bDone (BOOL)	Table transmitted (True) or it is not finalised (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

ExecuteMap

Examples:

Declaration:

```
INST: TransferMapData;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

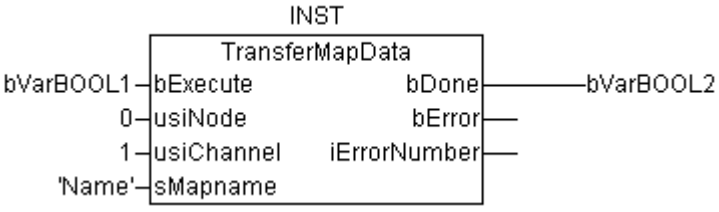
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, sMapname := 'Name')
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, sMapname := 'Name');
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



UnlinkSlaveToMaster

PMCprimo-Command: UL (**Un**link slave axis from master axis)

Function library: **Mapping**

Description:

With this function block the instantaneous slave axis becomes a single axis again. The definition of a slave axis must be cancelled, before the axis is allocated to another master axis.

Input variables:

bExecute (BOOL)	The definition is cancelled in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Definition cancelled (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

MapLinkSlaveToMaster and MapLinkSlaveToDifferentialMaster

Examples:

Declaration:

```
INST: UnlinkSlaveToMaster;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

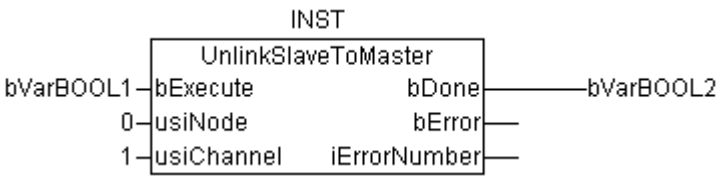
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



14.12 Output

DefineAnalogueLimitErrorOutput

PMCprimo-Command: AE (Define analogue limit error output)

Function library: **Output**

Description:

This function block defines a digital output of PMCprimo as error output. If the value of the analogue input exceeds one of the adjusted limit value, the error output changes its logical condition. The logical condition of the output is in case of an error this one, which has been specified with the sign ±. If the value of the analogue input returns to the permissible range, the status of the error output changes back again.

Input variables:

bExecute (BOOL)	The output is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	1 to 4 (3 and 4 is virtual)
usiOutput (USINT)	The output number from 1 to 8
usiPolarity (USINT)	The polarity 0 positive 1 negative

Output variables:

bDone (BOOL)	Output has been defined (True) or definition is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetAnalogueInputHighLimit and SetAnalogueInputLowLimit

The function block UndefineOutput is available for cancellation of the output definition.

Factory setting: no output defined

Examples

Declaration:

```
INST: DefineAnalogueLimitErrorOutput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

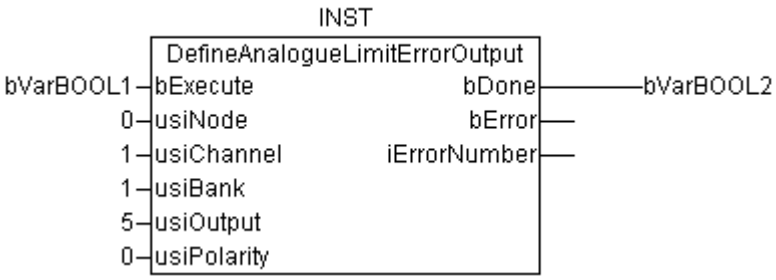
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



DefineBoundOverflowOutput

PMCprimo-Command: BO (Define **b**ound **o**verflow output)

Function library: **Output**

Description:

With this function block a pulse output can be activated in case of exceeding the position bound. If the selected axis exceeds the cycle limit, PMCprimo gives a pulse of 1ms to the defined output.

Input variables:

bExecute (BOOL)	The output is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	1 to 4 (3 and 4 is virtual)
usiOutput (USINT)	The output number from 1 to 8
usiPolarity (USINT)	The polarity 0 positive 1 negative

Output variables:

bDone (BOOL)	Output has been defined (True) or definition is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetPositionBound

The function block UndefineOutput is available for cancelling the output definition.

Factory setting: No output defined

Examples:

Declaration:

```
INST: DefineBoundOverflowOutput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

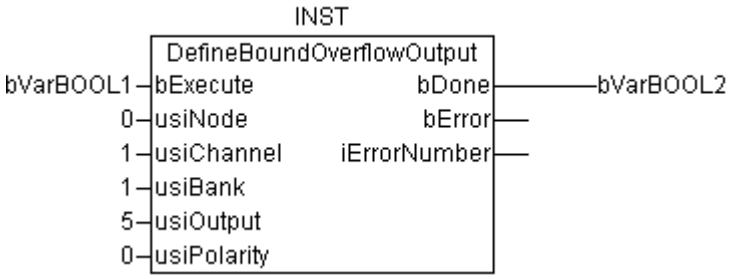
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



DefineMotorErrorOutput

PMCprimo-Command: DE (**D**efine **e**rror output)

Function library: **Output**

Description:

With this function block a digital output can be defined as error output. The specified output directs the specified signal level, if PMCprimo has detected an axis error (motor OFF). In case the position control loop is closed again (function block EnablePositionControl) the error output is reset.

In case of the following errors PMCprimo changes the signal level at a defined error output:

- **Following error detected (SetMaxPositionError)**
- **Monitoring time encoder signals exceeded (SetEncoderTimeout)**
- **Motor position outside the software limts (SetHighPositionLimit and SetLowPositionLimit)**
- **Errors, which can be released with SetErrorOptionsWord to an axis error**

Input variables:

bExecute (BOOL)	The output is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	1 to 4 (3 and 4 is virtual)
usiOutput (USINT)	The output number from 1 to 8
usiPolarity (USINT)	The polarity 0 positive 1 negative

Output variables:

bDone (BOOL)	Output has been defined (True) or definition is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetMaxPositionError, SetEncoderTimeout, SetHighPositionLimit, SetLowPositionLimit and SetErrorOptionsWord

The function block UndefineOutput is available for cancelling the output definition.

Factory setting: No output defined

Examples

Declaration:

```
INST: DefineMotorErrorOutput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

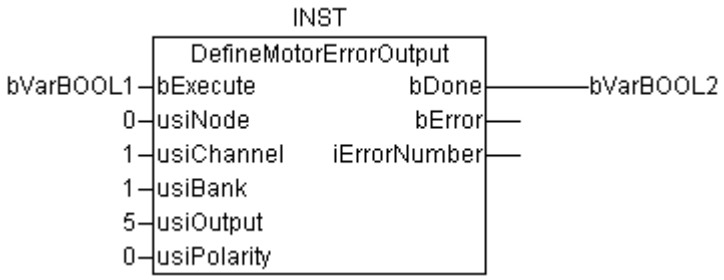
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



DefineOutsideWindowOutput

PMCprimo-Command: OW (Define **o**utside **w**indow output)

Function library: **Output**

Description:

This function block defines an output for watching position error. If the position error is bigger than SetWindow (page 200) the output is set/ reset. A line which has been defined as an outside window output may be returned to normal operation by entering this command without the sign.

Input variables:

bExecute (BOOL)	The output is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	1 to 4 (3 and 4 is virtual)
usiOutput (USINT)	The output number from 1 to 8
usiPolarity (USINT)	The polarity 0 positive 1 negative

Output variables:

bDone (BOOL)	Output has been defined (True) or definition is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetMaxPositionError, SetEncoderTimeout, SetHighPositionLimit, SetLowPositionLimit and SetErrorOptionsWord

The function block UndefineOutput is available for cancelling the output definition.

Factory setting: No output defined

Examples

Declaration:

```
INST: DefineOutsideWindowOutput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

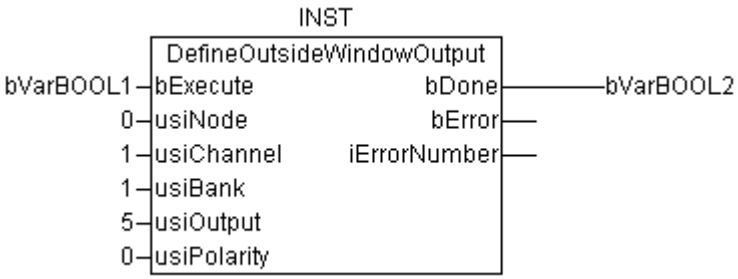
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



DefinePositionTriggerOutput

PMCprimo-Command: PO (Define **p**osition trigger **o**utput)

Function library: **Output**

Description:

With this function block a digital output can be defined as electronic cam. The specified output directs specified signal levels within the position range specified according to the sign.

If SetPositionBound is set to 1000 and a position trigger output is defined between 0 and –200 for example and it is moved with constant velocity in negative direction, there are different behaviour with different versions:

The output and the axis must be on the same network participant.

Input variables:

bExecute (BOOL)	The output is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	1 to 4 (3 and 4 is virtual)
usiOutput (USINT)	The output number from 1 to 8
usiPolarity (USINT)	The polarity 0 positive 1 negative
diPostion1 (DINT)	The initial position (+/- 4.000.000)
diPostion2 (DINT)	The final position (+/- 4.000.000)

Output variables:

bDone (BOOL)	Output has been defined (True) or definition is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetPositionBound, SetPositionOutputHysteresis and SetPhaseAdvanceFactor

The function block UndefineOutput is available for cancelling the output definition.

Factory setting: No output defined

Examples:

Declaration:

```
INST: DefineMotorErrorOutput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

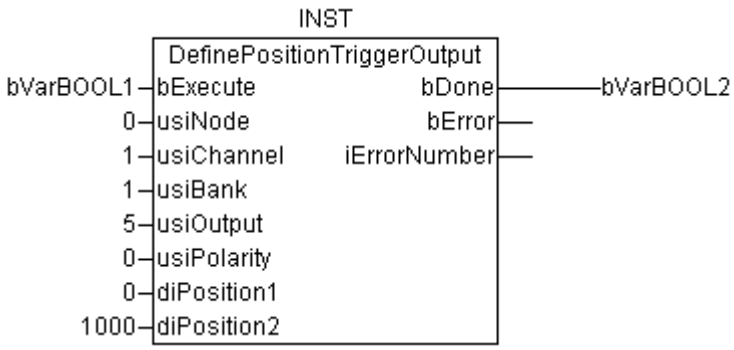
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0, diPosition1 := 0, diPosition2 := 1000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0, diPosition1 := 0, diPosition2 := 1000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



DefineReferenceAcceptedOutput

PMCprimo-Command: RA (Define **r**eference **a**ccepted output)

Function library: **Output**

Description:

With this function block a pulse output can be activated when recognising a valid reference input signal. If PMCprimo recognises the signal of a reference input as valid, PMCprimo gives a pulse of 1ms to the defined output. The polarity indicates the signal level of the output with the pulse output. This function can be used for example, to recognise products as ok, if the function blocks SetReferenceFalseHighLimit, SetReferenceFalseLowLimit, SetReferenceTrueHighLimit and SetReferenceTrueLowLimit (page 266) are used.

The pulse output is also executed during initialisation with InitialisePosition and InitialisePositionBounds.

Input variables:

bExecute (BOOL)	The output is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	1 to 4 (3 and 4 is virtual)
usiOutput (USINT)	The output number from 1 to 8
usiPolarity (USINT)	The polarity 0 positive 1 negative

Output variables:

bDone (BOOL)	Output has been defined (True) or definition is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetReferenceMode and DefineReferenceInput

The function block UndefineOutput is available for cancellation of the output definition.

Factory setting: no output defined

Examples:

Declaration:

```
INST: DefineReferenceAcceptOutput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

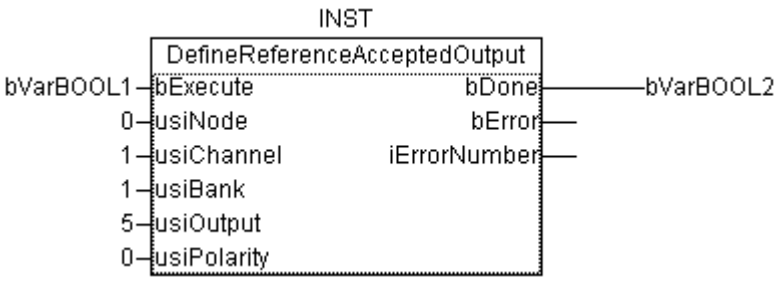
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



DefineReferenceBackwardOutput

PMCprimo-Command: JB (Reference ad**j**ustment **b**ackwards output)

Function library: **Output**

Description:

With this function block an output can be set for the time, for which a reference error correction is executed against the driving direction. This means that the axis must get slower in case of a reference error correction, so that the defined output is set to “true”.

Input variables:

bExecute (BOOL)	The output is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	1 to 4 (3 and 4 is virtual) 1 to 20, adjustable with CD command (PMCprimo 16+) 1 to 10, adjustable with CD command (PMCprimo Drive2)
usiOutput (USINT)	The output number from 1 to 8
usiPolarity (USINT)	The polarity 0 positive 1 negative

Output variables:

bDone (BOOL)	Output has been defined (True) or definition is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetReferenceMode, DefineReferenceForwardOutput, DefineReferenceOutput and DefineReferenceInput

The function block UndefineOutput is available for cancellation of the output definition.

Factory setting: no output defined

Examples:

Declaration:

```
INST: DefineReferenceBackwardOutput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

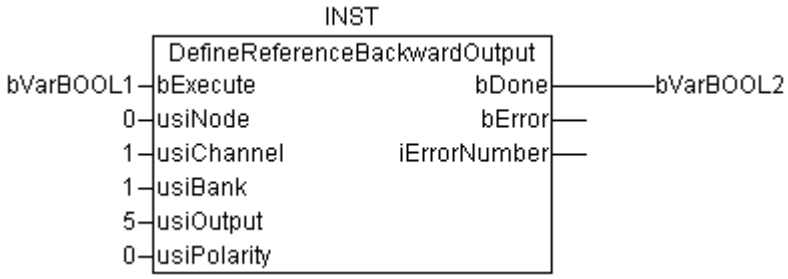
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



DefineReferenceForwardOutput

PMCprimo-Command: JF (Reference ad**j**ustment **f**orwards output)

Function library: **Output**

Description:

With this function block an output can be set for the time, for which a reference error correction is executed in the driving direction. This means that the axis must get faster in case of a reference error correction, so that the defined output is set to “true”.

Input variables:

bExecute (BOOL)	The output is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	1 to 4 (3 and 4 is virtual) 1 to 20, adjustable with CD command (PMCprimo 16+) 1 to 10, adjustable with CD command (PMCprimo Drive2)
usiOutput (USINT)	The output number from 1 to 8
usiPolarity (USINT)	The polarity 0 positive 1 negative

Output variables:

bDone (BOOL)	Output has been defined (True) or definition is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetReferenceMode, DefineReferenceBackwardOutput, DefineReferenceOutput and DefineReferenceInput

The function block UndefineOutput is available for cancellation of the output definition.

Factory setting: no output defined

Examples:

Declaration:

```
INST: DefineReferenceForwardOutput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

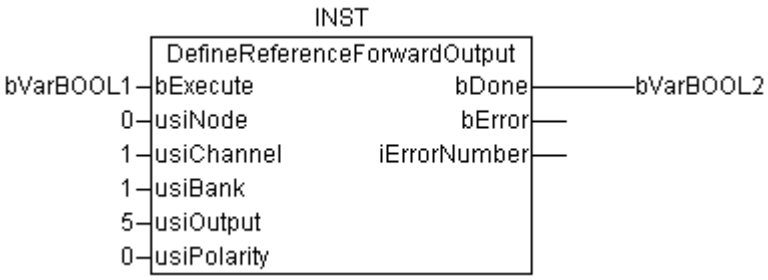
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



DefineReferenceOutput

PMCprimo-Command: OR (Set **r**eference ad**j**ustment output)

Function library: **Output**

Description:

With this function block an output can be set for the time to “true”, for which a reference error correction is executed. With the function blocks DefineReferenceBackwardOutput and DefineReferenceForwardOutput an output for a certain correcting direction is defined. With this function block an output signal is executed in both correcting directions.

Change as of **version 1.008a**:

The output is set for 1 millisecond to 1, also the reference error is 0. In older versions the output was not set in this case.

Input variables:

bExecute (BOOL)	The output is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	1 to 4 (3 and 4 is virtual)
usiOutput (USINT)	The output number from 1 to 8
usiPolarity (USINT)	The polarity 0 positive 1 negative

Output variables:

bDone (BOOL)	Output has been defined (True) or definition is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetReferenceMode, DefineReferenceBackwardOutput, DefineReferenceForwardOutput and DefineReferenceInput

The function block UndefineOutput is available for cancellation of the output definition.

Factory setting: no output defined

Examples:

Declaration:

```
INST: DefineReferenceOutput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

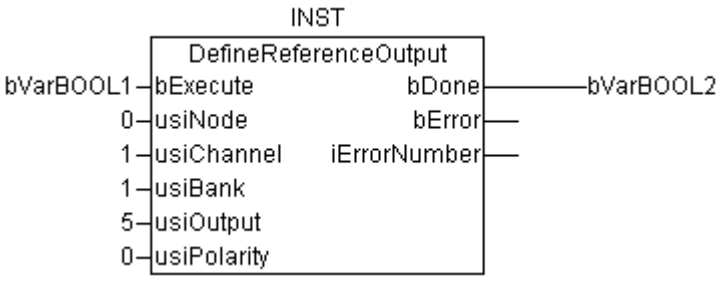
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



DefineReferenceRejectOutput

PMCprimo-Command: RR (Define **r**eference **r**eject output)

Function library: **Output**

Description:

With this function block a digital output can be defined as reference error output. A reference error is measured with each reference signal. The measured reference error is compared with the value of SetMaxReferenceCorrection. In case the measured reference error is within the specified limits, the reference error output is set as incorrectly. In the contrary case when the measured reference error exceeds the specified limits, the reference error output is set as true. The signal level of the reference error output is specified with the argument polarity. The signal level is kept so long, until a new comparison was executed with the next reference signal.

A typical application for this function is, for example, the control of a distribution point which distinguishes between good and worse elements. For example, the front edge of elements lying on a conveyer belt is measured with a photo cell. The measured value in connection with the value of SetMaxReferenceCorrection can be used for the control of a distribution point to disconnect good and worse elements.

Input variables:

bExecute (BOOL)	The output is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	1 to 4 (3 and 4 is virtual)
usiOutput (USINT)	The output number from 1 to 8
usiPolarity (USINT)	The polarity 0 positive 1 negative

Output variables:

bDone (BOOL)	Output has been defined (True) or definition is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetReferenceMode and DefineReferenceInput

The function block UndefineOutput is available for cancellation of the output definition.

Factory setting: no output defined

Examples:

Declaration:

```
INST: DefineReferenceRejectOutput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

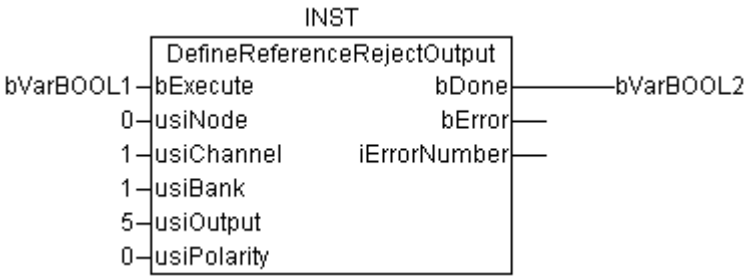
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



DefineTimerOutput

PMCprimo-command: TC (Define **t**imer/**c**ounter output)

Function library: **Output**

This function is available since Version 2.005.

Description:

With this command it is possible to define a digital output for timer or counter function. If for example a bound overflow output is defined it is possible extend the implulse with “DefineTimerOutput”, in order CoDeSys can detect it certainly.

The timer/counter can only be triggerd from PMCprimo.

The timer/counter mode values are as follows:

Mode	Operation
0	One-shot up counter
1	Cyclic up counter
2	One-shot down counter
3	Cyclic down counter
4	One-shot timer
5	Cyclic timer
6	One-Shot-Timer with restart
7	Reserved
8	Reserved

The output line is set true (as defined with usiPolarity) when the timer/counter is first triggered, and it is reset false when the timer/counter reaches its final value, set by the count parameter.

In all counter modes, the counter is incremented or decremented when either the output is set or reset by PMCprimo. If it is an up counter, its initial value is zero, and its final value is given by the count parameter. If it is a down counter, its initial value is the count parameter, and its final value is zero. On the first count, the output line is set to true, and the counter is started and set to its initial value. When the count is incremented or decremented to its final value, the output line is reset false, and the counter is stopped and reset to its initial value. The counter can only activated

by a outputdefinition of PMCprimo.

In timer modes the timer is triggered in the same way as in counter mode, but once triggered it counts once per tick until the final count is reached. The output line is set true when the timer is triggered, and is reset false when the timer reaches the final count.

In one-shot modes, the timer/counter behaves as described above. In cyclic modes it operates in a slightly different way. When the timer counter reaches its final count, the output line is toggled to its opposite state, the counter/timer is reset to its initial value and continues to run. The output line changes state each time the counter/timer reaches its final count.

Input variables:

bExecute (BOOL)	The output is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiBank (USINT)	1 to 4 (3 and 4 is virtual)
usiOutput (USINT)	The output number from 1 to 8
usiPolarity (USINT)	The polarity 0 positive 1 negative
diValue (DINT)	Value for timerfunction 0 bis 65535
usiModus (USINT)	Modus of Timerfunction (see above)

Output variables:

bDone (BOOL)	Output has been defined (True) or definition is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

DefineBoundOverflowOutput and DefinePositionTriggerOutput

The function block UndefineOutput is available for cancellation of the output definition.

Factory setting: no timer defined

Example:

Declaration:

```
INST: DefineTimerOutput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

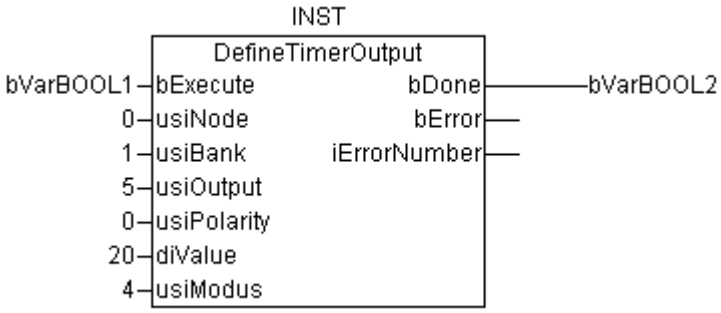
Example in AWL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, , usiBank := 1, usiOutput := 5
            , usiPolarity := 0, diValue :=20, usiModus :=4)
LD    INST.bDone
ST    bVarBOOL2
```

Example Beispiel in ST:

```
NST(bExecute := bVarBOOL1, usiNode := 0, , usiBank := 1, usiOutput := 5
            , usiPolarity := 0, diValue :=20, usiModus :=4);
bVarBOOL2:=Inst.bDone;
```

Example in FUP:



DefineVelocityOutput

PMCprimo-Command: VO (Define **v**elocity trigger **o**utput)

Function library: **Output**

Description:

With this function block a digital output can be set dependent on the momentary velocity of an axis. The specified output directs the specified signal level within the velocity range specified according to the sign. With the function block SetVelocityOutputHysteresis a hysteresis can be set, so that the output does not switch constantly, if the actual velocity varies by the switch thresholds.

Input variables:

bExecute (BOOL)	The output is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	1 to 4 (3 and 4 is virtual)
usiOutput (USINT)	The output number from 1 to 8
usiPolarity (USINT)	The polarity 0 positive 1 negative
diSpeed1 (DINT)	The initial speed of the range
diSpeed2 (DINT)	The final speed of the range

Output variables:

bDone (BOOL)	Output has been defined (True) or definition is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetVelocityOutputHysteresis

The function block UndefineOutput is available for cancellation of the output definition.

Factory setting: no output defined

Examples:

Declaration:

```
INST: DefineVelocityOutput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

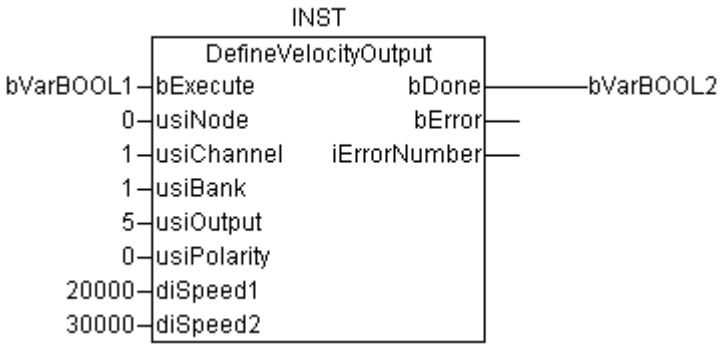
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0, diSpeed1 := 20000, diSpeed2 := 30000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0, diSpeed1 := 20000, diSpeed2 := 30000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetPhaseAdvanceFactor

PMCprimo-Command: PA (Set **p**hase **a**dvance scale factor)

Function library: **Output**

Description:

With this function block an electronic cam of an axis can be provided with a phase shifting depending on the velocity. This phase shifting can be arranged separately for each output.

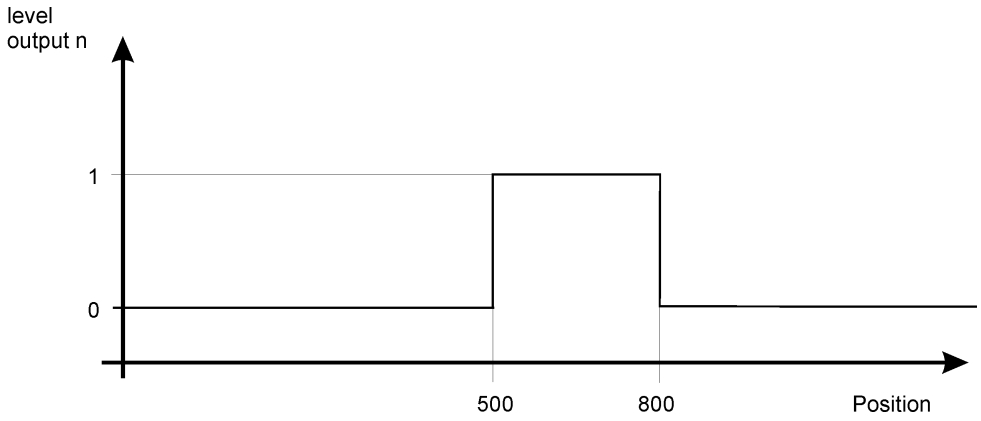
Phase shifting output = $(^{velocity}/_{256}) \times (^{PA}/_{256})$

The velocity of the axis is indicated in increments/second, for example a phase shifting of 153 increments is obtained with a velocity of 20.000 increments/seconds and a value of 500.

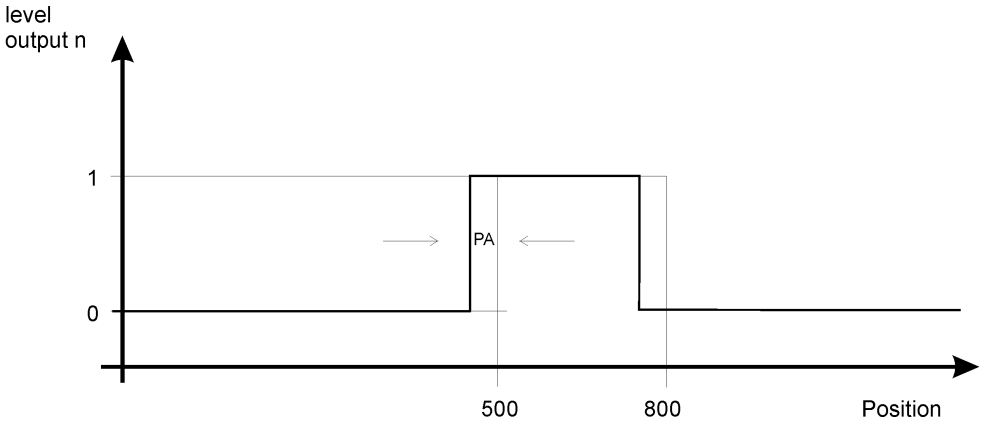
This factor can also be considered as set time. A value of 1 corresponds to 15,625 microseconds. A phase shifting of 64, therefore, corresponds to 1 millisecond and, therefore, to the way, this axis moves in this time. Is the delay in time, for example of the connected valve, known, the factor can be calculated quite easily.

For the calculation the averaging actual velocity is used. The time base can be set with the function block SetVelocityAveragingTime.

Electric cam without speed depending phase shift:



Electric cam with speed depending phase shift at a speed of v = 1000 inc/s and a phase shift SetPhaseAdvanceFactor = 2560:



Electric cam with speed depending phase shift at a speed of v = 5000 inc/s and a phase shift SetPhaseAdvanceFactor = 2560:

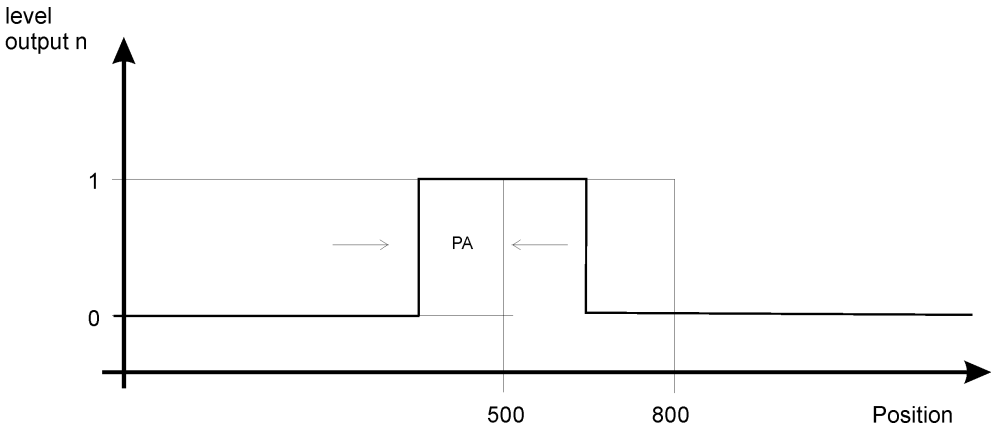


Illustration 19: Speed depending phase shift

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	1 to 4 (3 and 4 is virtual)
usiOutput (USINT)	The output number from 1 to 8
usiPolarity (USINT)	The polarity 0 positive 1 negative

Output variables:

bDone (BOOL)	Output has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

DefinePositionTriggerOutput and SetVelocityAveragingTime

Factory setting: 0 (switched-off)

Examples

Declaration:

```
INST: SetPhaseAdvanceFactor;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

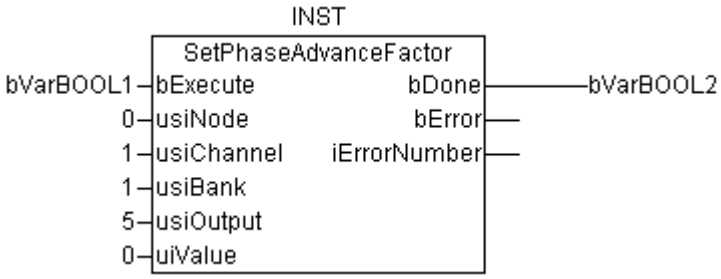
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, usiBank := 1
            , usiOutput := 5, usiPolarity := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetPositionOutputHysteresis

PMCprimo-Command: PH (Position Output Hysteresis)

Function library: **Output**

Description:

This function block can be used for an extension of the function block DefinePositionTriggerOutput. A hysteresis is added to the switching points, hence the output is not constantly switched on and off during stoppage, if for example the actual position varies by 1 increment. The set value is either added to the cam position or subtracted from it.

The value can be used for all outputs on this axis.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiValue (UINT)	The hysteresis for all cams on this axis

Ab **Primo_V2_006.lib**: udiValue (UDINT)

Output variables:

bDone (BOOL)	Output has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

DefinePositionTriggerOutput

Factory setting: 0 (switched off)

Examples:

Declaration:

```
INST: SetPositionOutputHysteresis;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

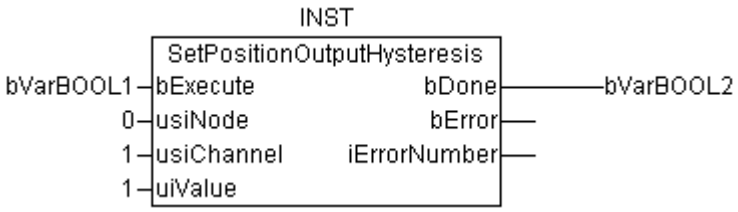
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue := 1 )
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, uiValue := 1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetVelocityOutputHysteresis

PMCprimo-Command: VH (Velocity Output Hysteresis)

Function library: **Output**

Description:

This function block is used for an extension of the function block DefineVelocityOutput. A hysteresis is added to the switching points, hence the output is not constantly switched on and off. The set value is either added to the set switching points of the velocity or subtracted from them.

The value is used for all outputs of the axis.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiValue (UINT)	The hysteresis for all velocity outputs on this axis

As of **Primo_V2_006.lib**: udiValue (UDINT)

Output variables:

bDone (BOOL)	Output has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

DefineVelocityOutput

Factory setting: 0 (switched-off)

Examples:

Declaration:

```
INST: SetVelocityOutputHysteresis;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

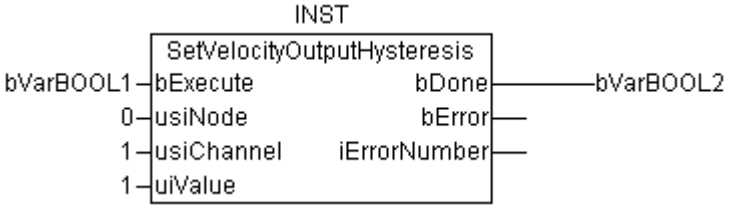
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue := 1 )
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute:=bVarBOOL1, usiNode:=0, usiChannel:=1, uiValue := 1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



UndefineOutput

PMCprimo-Command: UO (**U**ndefine **o**utput definition)

Function library: **Output**

Description:

With this function block the user cancels a defined output function in PMCprimo independent which function was defined before.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	1 to 2, PMCprimo Drive/2, (2 is virtual) 1 to 3, PMCprimo 2+2, (3 is virtual) 1 to 3 (PMCprimo 16+) (3 is virtual)
usiOutput (USINT)	The output number from 1 to 8

Output variables:

bDone (BOOL)	Definition was cancelled (True) or function block is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

DefineAnalogueLimitErrorOutput, DefineAuxiliaryOutput, DefineBoundOverflowOutput, DefineMotorErrorOutput, DefinePositionTriggerOutput, DefineReferenceAcceptedOutput, DefineReferenceBackwardOutput, DefineReferenceForwardOutput, DefineReferenceOutput, DefineReferenceRejectOutput und DefineVelocityOutput

Examples:

Declaration:

```
INST: UndefineOutput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

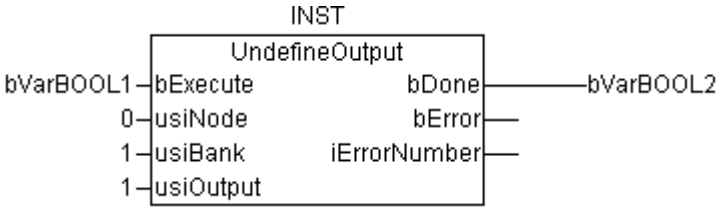
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0,  usiBank := 1, usiOutput := 5)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0,  usiBank := 1, usiOutput := 5);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



14.13 Positioncontrol

The drive is operated with all position processes in a position control. In case there are deviations between an actual position and the constant or consecutive setpoint positions, these ones are reduced by the position controller.

Control algorithm:

PMCprimo includes a PID-position controller with reverse, velocity and acceleration pre-control factors:

$$V_{Demand} = KP\ e_i + KI \sum e_i + KD\ (e_i - e_{i-1}) + KV\ (p_i - p_{i-1}) + KF(d_i - d_{i-1}) + KA[(d_i - d_{i-1}) - (d_{i-1} - d_{i-2})]$$

KP = proportional gain constant
KI = integral gain constant
KD = differential gain constant
KV = velocity feedback gain constant
KF = velocity feed-forward gain constant
KA = acceleration feed-forward gain constant
e_i = position error (= demand position - measured position)
d_i = demand position
p_i = measured position

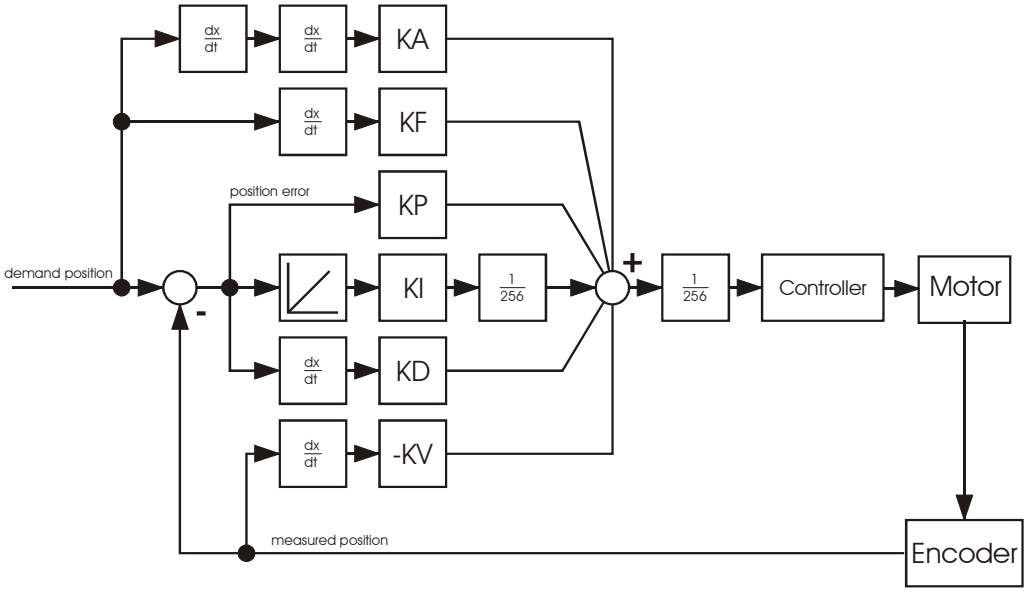
The dynamic behaviour of the drive depends on these constant factors and on the mechanical behaviour of the driven machines. The setting of these factors is essential for obtaining an optimum control behaviour.

The following connection between the following error and setpoint issue is valid:

$$V_{Soll} = \text{error} * KP/256 * 10/2048$$

PMCprimo offers additional functions for tuning the control loop. At the analogue output (e.g. master axis is not position controlled) different information can be shown at an oscilloscope or at another recording unit

Block circuit diagram of the position control:



EnablePositionControl

PMCprimo-Command: PC (Enter **p**osition **c**ontrol mode)

Function library: **Positioncontrol**

Description:

This function block activates the position control of the motor.

In case of an active position control loop a relay per axis is controlled with the PMCprimo 2+2 with which the control unit is released for the drive.

The position control is deactivated in case of a motor error. The active position control signals PMCprimo with the sign „ > “ at the serial interface or at the LED Display (signal see installation manual).

Input variables:

bExecute (BOOL)	The position control is activated in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Position control is active (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function::

MotorOff und GlobalOff

Factory setting: Release switched-off

Examples:

Declaration:

```
INST: EnablePositionControl;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

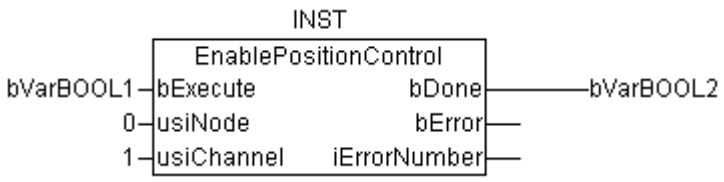
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0,  usiChannel := 1,
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0,  usiChannel := 1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



GlobalOff

PMCprimo-Command: GF (Global motor off)

Function library: **Positioncontrol**

Description:

All motors are switched-off. This function block is an all axis global MotorOff (page 177)

Input variables:

bExecute (BOOL)	All axes are switched off in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0

Output variables:

bDone (BOOL)	All motors are switched-off (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

MotorOff

Examples:

Declaration:

```
INST: GlobalOff;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

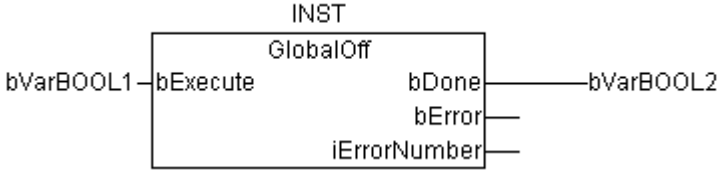
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



GlobalStop

PMCprimo-Command: GS (Global stop)

Function library: **Positioncontrol**

Description:

All axes decelerate with their corresponding ramp given by SetDeceleration. This function block is an all axes global StopMotor -Function (page 242).

Input variables:

bExecute (BOOL)	All axes are stopped in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0

Output variables:

bDone (BOOL)	All axes are stopped (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function

StopMotor

Examples:

Declaration:

```
INST: GlobalStop;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

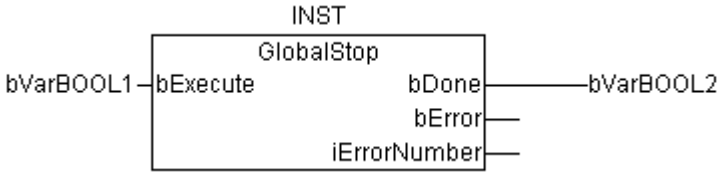
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



InitialiseDemandOffset

PMCprimo-Command: ID (Initialise Demand Offset)

Function library: **Positioncontrol**

As of **version 2.004** available

Description:

Under normal conditions, there may be some constant offset in the demand signal analogue output amplifiers which causes the motor to settle at a position slightly different to the required position. The "**InitialiseDemandOffset**" command sets the system up to correct for this (assumed constant) offset in all subsequent position control operations. It must be used every time the system is powered on, when the system is in the position control mode, to set the actual position as close as possible to the required position. This is particularly necessary when the final position outside the final position window, and at the end of a move command it returns the error message .failed to reach target position. The "**InitialiseDemandOffset**" command actually controlling the position, and it has no effect if the motor is not driving the system. Note that friction in the mechanical system can also cause a position offset after a move command is executed.

bExecute (BOOL)	The function is started in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Motor is switched off (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Examples:

Declaration:

```
INST: InitialiseDemandOffset;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

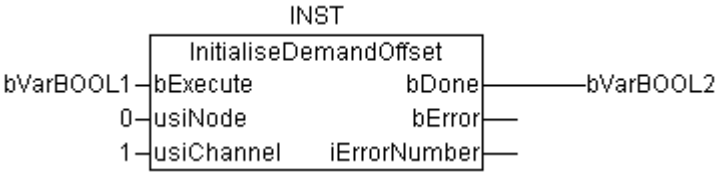
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0,  usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0,  usiChannel := 1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



MotorOff

PMCprimo-Command: MO (**M**otor **o**ff)

Function library: **Positioncontrol**

Description:

This function block switches off the controller release. All other functions remain active, the encoder signals are evaluated. In case the controller release is re-activated, the motor remains in the instantaneous position.

In case of an open position control loop the setpoint signal is damped with 0V with the PMCprimo 2+2/16+ and the relay of the axis opens. This relay should be used as controller or drive release. The relay simultaneously damps the setpoint output with 0V. In case the motor controller is locked with the relay, the motor coasts freely.
This function can also be used as alternative Stop-Command. With the function GlobalOff (see page 177) the function MotorOff can be executed at all axes at the same time.
Input variables:

bExecute (BOOL)	The motor is switched off in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Motor is switched off (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function

MotorOff und GlobalOff

Examples:

Declaration:

```
INST: MotorOff;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

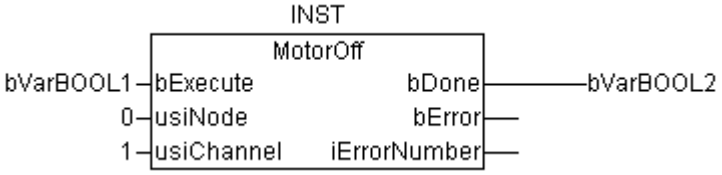
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0,  usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0,  usiChannel := 1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetAccelerationFeedForwardGain

PMCprimo-Command: KA (Set acceleration feed-forward gain constant)

Function library: **Positioncontrol**

Description:

With this function block the acceleration factor of the control loop algorithm is set. The factor uses the set acceleration contrary to the actual acceleration and helps if the axis is driven with a very high acceleration. The following error can be minimised in the acceleration phase by the aid of the factor.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiGain (UINT)	The control factor from 0 to 65535

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function

SetProportionalGain,SetDifferentialGain, SetIntegralGain, SetVelocityFeedbackGain und SetVelocityFeedForwardGain

Factory setting: 0 (switched off)

Examples:

Declaration:

```
INST: SetAccelerationFeedForwardGain;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

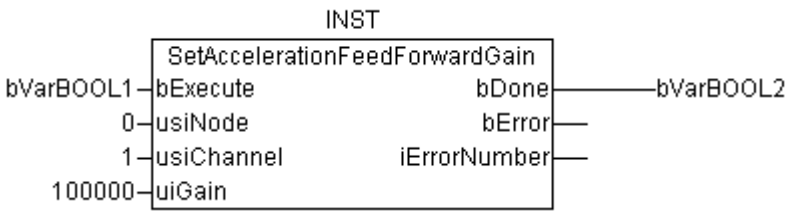
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, usiNode := 0,  usiChannel := 1,  uiGain := 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0,  usiChannel := 1,  uiGain := 1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetControlWord

PMCprimo-Command: CW (Set control word)

Function library: **Positioncontrol**

Description:

With this function block the user can determine the counting direction of the encoder, the velocity curve for moving commands and the sign of the setpoint output of each axis (Bit 0 stands with inputs and outputs on the right and Bit 7 on the left).

Attention: The counting direction of the encoder and the sign of the setpoint output should only be modified with an open position control loop. This inverting functions are used for simplification of the initial start-up of the motor (saves rewiring) and allows an adaptation of the direction of motion of PMCprimo to the definition of the direction of motion of the user.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
usiControlword (USINT)	The control word with following meaning.

Bit 0: Program abort with a motor error:

0: Axle programs which wait for an execution of a moving command are stopped.

1: Axle programs are not stopped due to motor errors.

Bit 1: Storing of positions in the battery backed SRAM. This function is only available in the PMCprimo 2+2/16+.

0: Positions are not stored in SRAM.

1: Actual position is permanently kept in SRAM. This Bit cannot be stored. If the last position shall be read in by SRAM after switching-on, the Bit must be set in the start-up program. The data are reflected in SRAM and checked after switching-on. In case of a failure an error message is signalled axle specific.

Attention: This bit may only set, when all commands affecting the position are set (command “**SetFeedbackEncoder**“, “**SetNumberOfBits**“, “**SetEncoderScaling**“, “**SetPositionBound**“). If the bit is set too early, then the position in the ram may be wrong.

Enhancement as of version **2.004**:
If a motor is moved to a negative position and then PMCprimo is switched off and on then the position is calculated to a negative position. In former versions the position was calculated to a positive position. It is important that the motor is not moved more than the half bound length manually when the system is switched off.

Bit 2: Definition of the velocity curve for accelerations and braking actions.

0: Trapezoidal velocity curve for the functions MoveToAbsolutePosition, MoveRelativePosition, MoveConstantVelocity and StopMotor

1: Sinusoidal velocity curve for the commands functions MoveToAbsolutePosition, MoveRelativePosition, MoveConstantVelocity and StopMotor

Bit 3: Behaviour in case of motor errors:

0: Axis is immediately switched-off (MotorOff)

1: Axis brakes with the brake ramp and switches-off the motor.

Bit 4: Defines the sign setpoint output.

0: Setpoint sign normal, i.e. if the encoder counts in a positive direction, the sign of the setpoint is negative.

1: Setpoint sign inverted, i.e. if the encoder counts in a positive direction, the sign of the setpoint is also positive.

Bit 5: Defines the counting direction of the encoder.

0: Counting direction normal, i.e. if track B follows on track A, the position counter of PMCprimo counts in a positive direction.

1: Counting direction inverted, i.e. of track B follows on track A the position counter of PMCprimo counts in a negative direction.

Bit 6: Function SetEncoderTimeout for encoder monitoring.

0: SetEncoderTimeout defines a monitoring time.

1: SetEncoderTimeout defines a way for monitoring.

Bit 7: Defines, if the factor of the function SetIntegralGain is always or only during standstill active.

0: The factor is always active.

1: The factor is only active in case of a standing drive.

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetEncoderTimeout and SetIntegralGain

Factory setting: 16 (= 10hex)

Examples:

Declaration:

```
INST: SetControlWord;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

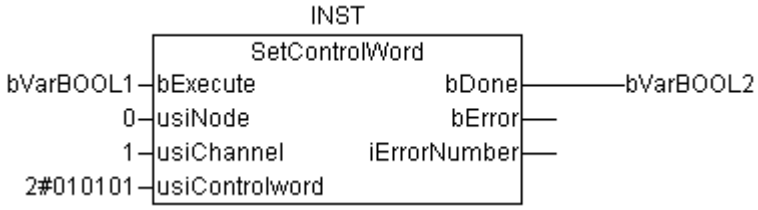
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1,usiControlword := 2#010101)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiControlword := 2#010101);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetProportionalGain

PMCprimo-Command: KP (Set **p**roportional gain constant)

Function library: **Positioncontrol**

Description:

With this function block the proportional factor of the control algorithm is set. A high value of the KP-factor allows a short reaction time and an exact position control. Therefore, the KP-factor should be set as high as possible without generating an overshooting.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
uiGain (UINT)	The control factor from 0 to 65535

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetAccelerationFeedForwardGain , SetDifferentialGain, SetIntegralGain, SetVelocityFeedbackGain und SetVelocityFeedForwardGain

Factory setting: 10

Examples:

Declaration:

```
INST: SetProportionalGain;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

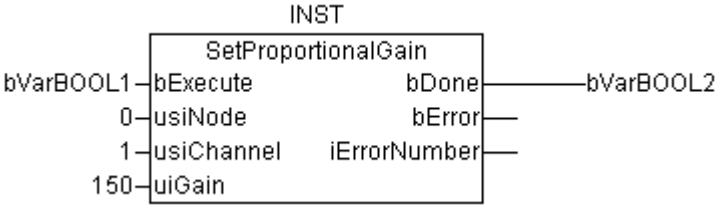
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiGain := 150)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiGain := 150);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetDifferentialGain

PMCprimo-Command: KD (Set **d**ifferential gain constant)

Function library: **Positioncontrol**

Description:

With this function block the differential factor of the control algorithm is set. The differential factor is useful with very strong varying position errors, e.g. with a step-by-step position modification. The factor has only a secondary meaning in a well tuned control loop.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0.
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
uiGain (USINT)	The control factor from 0 to 65535

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetProportionalGain,SetAccelerationFeedForwardGain , SetIntegralGain, SetVelocityFeedbackGain und SetVelocityFeedForwardGain

Factory setting: 0 (switched-off)

Examples:

Declaration:

```
INST: SetDifferentialGain;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

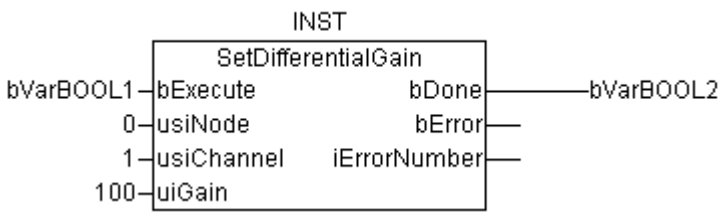
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiGain := 100)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiGain := 100);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetIntegralGain

PMCprimo-Command: KI (Set **i**ntegral gain constant)

Function library: **Positioncontrol**

Description:

With this function block the integral factor of the control algorithm is set. The integral factor is useful in order to compensate a constant position error due to a continuous load and oscillations or with a velocity control. The integral factor ,however, also effects that the axis goes beyond the target position in case of a positioning, as the position error, which arises during a movement, can accumulate. This problem is also known as >wind-up<.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
uiGain (UINT)	The control factor from 0 to 65535

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetProportionalGain,SetAccelerationFeedForwardGain , SetDifferentialGain, SetVelocityFeedbackGain und SetVelocityFeedForwardGain

Factory setting: 0 (switched off)

Examples:

Declaration:

```
INST: SetIntegralGain;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

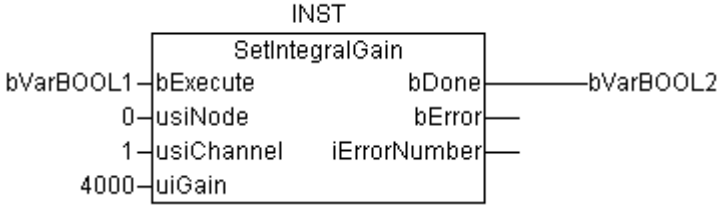
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiGain := 4000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiGain := 4000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetErrorOptionsWord

PMCprimo-Command: EW (Set **e**rror options **w**ord)

Function library: **Positioncontrol**

Description:

With this function block the user can determine the behaviour of PMCprimo in case of an error (Bit 0 stands for input or output on the right and Bit 7 on the left).

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
usiControlword (USINT)	The error options word with following meaning:

Bit 0: Reaction of PMCprimo when exceeding the monitoring time of reference inputs (see also SetReferenceErrorLimit page 280).

0: In case of an error PMCprimo gives a signal.

1: In case of an error PMCprimo gives a signal and switches-off the controller release.

Bit 1: Reaction of PMCprimo when recognising a reference input outside the permissible range (see also SetReferenceTimeout page 262).

0: In case of an error PMCprimo gives a signal.

1: In case of an error PMCprimo gives a signal and switches-off the controller release.

Bit 2: Reaction of PMCprimo when recognising a reference signal before an existing reference error has been adjusted.

0: In case of an error PMCprimo gives a signal.

1: In case of an error PMCprimo gives a signal and switches-off the controller release

Bit 3: Reaction of PMCprimo when exceeding the maximum permissible limit values at the analogue input.

0: In case of an error PMCprimo gives a signal.

1: In case of an error PMCprimo gives a signal and switches-off the controller release

Bit 4: Suppress the error message „reference input is missing” (see also SetReferenceTimeout page 280), if Bit 0 is not set to 1.

0: The error message is indicated.

1: The error message is not indicated, if Bit 0 is set to 0.

Bit 5: Suppress the error message „reference error exceeded” (see also SetReferenceErrorLimit page 262), if Bit 1 is not set to 1.

0: The error message is indicated.

1: The error message is not indicated, if Bit 0 is set to 0..

Bit 6: Suppress the error message “reference signal recognised, before the existing reference error has been completely tuned”, if Bit 2 is not set to 1.

0: The error message is indicated.

1: The error message is not indicated, if Bit 0 is set to 0.

Bit 7: Suppress the error message “analogue input limit exceeded” (see also SetAnalogueInputHighLimit, SetAnalogueInputLowLimit page 296), if Bit 3 if not set to 1.

0: The error message is indicated.

1: The error message is not indicated.

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetReferenceTimeout and SetReferenceErrorLimit

Factory setting: 0

Examples:

Declaration:

```
INST: SetErrorOptionsWord;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

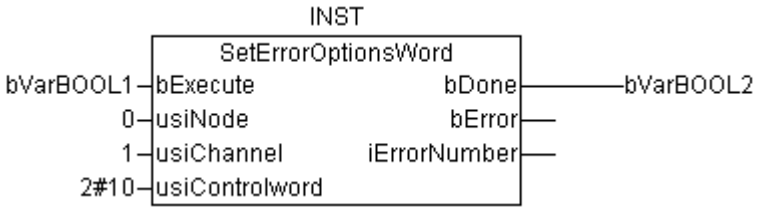
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, usiControlword := 2#10)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiControlword := 2#10);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetHighPositionLimit

PMCprimo-Command: LH (Set position **l**imit **h**igh)

Function library: **Positioncontrol**

Description:

This function block sets up a user-defined limit position. If at any time the absolute position of the motor exceeds the high position limit, PMCprimo gives the „high position limit exceeded“ error message and goes to the motor off state. This is similar to the action taken on detecting a limit switch input. The value is defined in encoder counts. If the SetPositionBound value is less than the high limit, then the high position limit checking is disabled, as the absolute position value wraps around to zero at the bound position before reaching the high limit position.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
diValue (DINT)	The software-end position +/- 4.000.000 increments As of version **2.004** new range: ±2.000.000.000

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetLowPositionLimit

Factory setting: 4.000.000 increments

Examples:

Declaration:

```
INST: SetHighPositionLimit;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

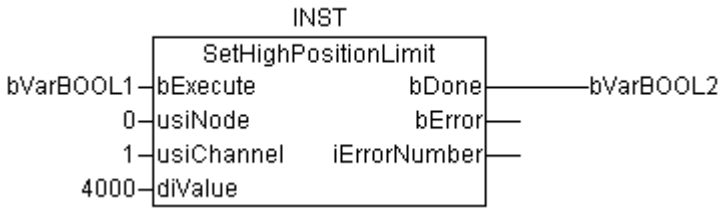
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, diValue := 4000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, diValue := 4000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetLowPositionLimit

PMCprimo-Command: LL (Set position **l**imit **l**ow)

Function library: **Positioncontrol**

Description:

With this function block a software-end position is set in a negative direction. PMCprimo only starts a positioning movement if the target position is within the software-end position. In case the software-end position is achieved with an end positioning, PMCprimo switches-off the controller release (motor is de-energised). The input of the software-end position is made in increments.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
diValue (DINT)	The software-end position +/- 4.000.000 increments As of version **2.004** new range: ±2.000.000.000

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function

SetHighPositionLimit

Factory setting: -4.000.000 increments

Examples:

Declaration:

```
INST: SetLowPositionLimit;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

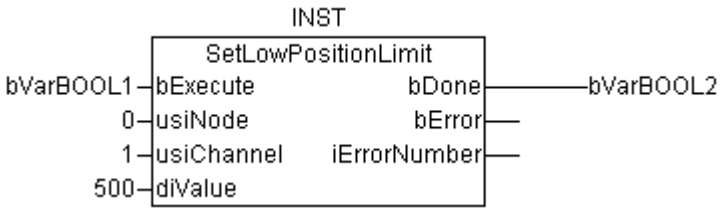
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, diValue := 500)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, diValue := 500);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetMaxPositionError

PMCprimo-Command: SE (**S**et maximum position **e**rror)

Function library: **Positioncontrol**

Description:

With this function block a maximum position error, which is continuously monitored by PMCprimo, is set. If the set error limit is exceeded PMCprimo brakes the axis up to the standstill and switches off the controller release (motor is de-energised). The input of the following error limit is made in increments.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
uiValue (UINT)	The maximum following error from 0 to 65535 increments

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function

SetControlWord

Factory setting: 10.000 increments

Examples:

Declaration:

```
INST: SetMaxPositionError;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

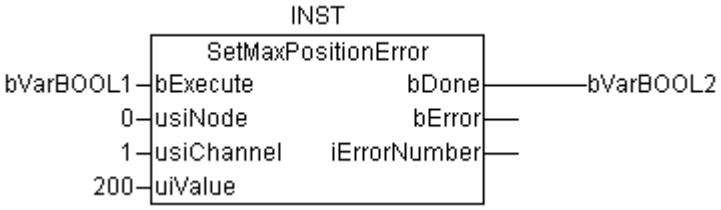
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiValue := 200)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue := 200);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetPositionCounter

PMCprimo-Command: ZC (**Z**ero position **c**ounters or set position)

Function library: **Positioncontrol**

Description:

The function block sets the position counter to the indicated value. In case the indicated value is bigger than the position bound (Function SetPositionBound) the position is scaled to the position bound.

Input variables:

bExecute (BOOL)	The position counter is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
diValue (DINT)	The new position in the range of +/- 4.000.000 increments

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetPositionBound

Examples:

Declaration:

```
INST: SetPositionCounter;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

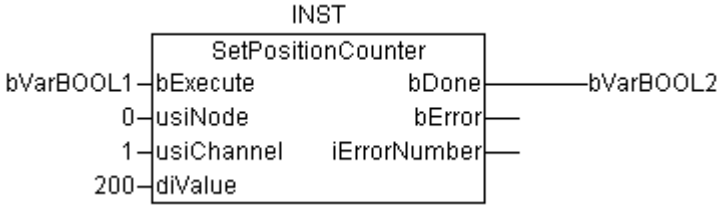
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, diValue := 200)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, diValue := 200);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetTimeoutForWindow

PMCprimo-Command: TiI (Set **TI**meout for Window)

Function library: **Positioncontrol**

As of **version 2.004** available

Description:

The check of SetWindow is made after the time SetTimeoutForWindow. The commands MoveToAbsolutePosition and MoveRelativePosition are after the adjusted time ready.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
uiValue (UINT)	The timeout from 0 to 65535 ms

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetWindow, MoveToAbsolutePosition und MoveRelativePosition

Factory setting: 0 (switched-of)f

Examples

Declaration:

```
INST: SetTimeoutForWindow;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiValue:= 100)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue:= 100);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:
Fehler! Es ist nicht möglich, durch die Bearbeitung von Feldfunktionen Objekte zu erstellen.

SetVelocityFeedbackGain

PMCprimo-Command: KV (Set **v**elocity feedback gain constant)

Function library: **Positioncontrol**

Description:

With this function block the actual value factor of the control algorithm is set. The actual value factor scales the motor velocity resulting from the measured position. The use of the factor can be compared with the effect from a tachometer sensor. The factor has a damping effect and allows a higher KP-factor and improves the reaction velocity of the axis, therefore.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
uiGain (UINT)	The control factor from 0 to 65535

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetAccelerationFeedForwardGain , SetDifferentialGain, SetIntegralGain, SetProportionalGain und SetVelocityFeedForwardGain

Factory setting: 0 (switched-off

Examples

Declaration:

```
INST: SetVelocityFeedbackGain;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

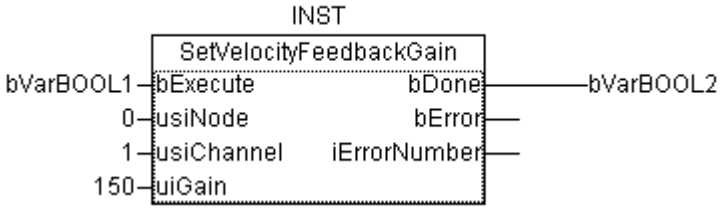
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiGain := 150)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiGain := 150);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetVelocityFeedForwardGain

PMCprimo-Command: KF (Set velocity feed-**f**orward gain constant)

Function library: **Positioncontrol**

Description:

With this function block the setpoint factor of the control algorithm is set. The setpoint factor influences the difference between the set velocity and the measured actual velocity. In case of a pure proportional controller a constant following error can be obtained during driving with constant velocity. By specifying a factor the following error can be reduced against zero or even brought to the negative range. The –factor is added to the required velocity so that the measured actual velocity is equal to the calculated setpoint velocity.

For the 'PMCprimo Drive' the value can be determined by the following formula:
KF = 15360 * 1000 / resolution per revolution

Example: resolution 4096 increments/revolution -> KF = 3750

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
uiGain (UINT)	The control factor from 0 to 65535

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetAccelerationFeedForwardGain , SetDifferentialGain, SetIntegralGain, SetProportionalGain und SetVelocityFeedbackGain

Factory setting: 0 (switched-off)

Examples:

Declaration:

```
INST: SetVelocityFeedForwardGain;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

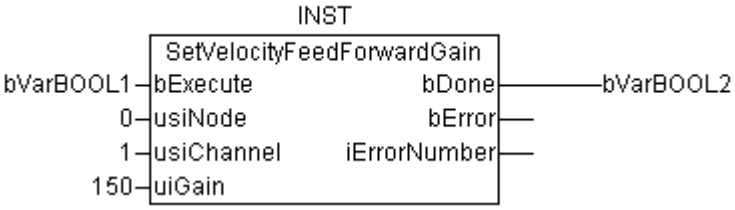
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiGain := 150)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiGain := 150);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetVirtualMotorMode

PMCprimo-Command: VM (Set virtual motor mode)

Function library: **Positioncontrol**

Description:

This function block is used in order to declare an axis to a virtual axis. In a virtual mode an axis acts without motor and encoder, as the actual position is internal calculated from the demand position.

Application of the virtual mode:

- Test of functions and programs without being connected to the machine.
- Simulation of a master axis.

Input variables:

bExecute (BOOL)	The motor is set in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiMode (USINT)	0: The axis is a real axis. A motor is operated with a position control and the encoder input is the actual value acceptance signal. 1: The axis is a virtual axis. The drive is not released by the axis. The setpoint signal is not defined and, therefore, a motor should not be connected to a virtual axis. sein. The actual value acceptance signal is internally simulated.

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Factory setting: 0 (Axis real)

Examples:

Declaration:

```
INST: SetVirtualMotorMode;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

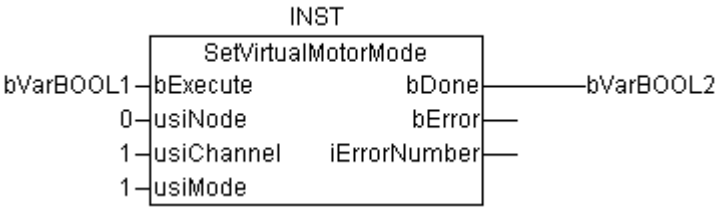
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, usiMode := 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiMode := 1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetWindow

PMCprimo-Command: SW (Set window)

Function library: **Positioncontrol**

Description:

This function block is used to specify the permissible target window in increments. PMCprimo signals position achieved, if the motor is within the window target after the braking action.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
uiWindow (UINT)	The target window from 0 to 65535 increments

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

MoveRelativePosition and MoveToAbsolutePosition

Factory setting: 100 increments

Examples:

Declaration:

```
INST: SetWindow;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

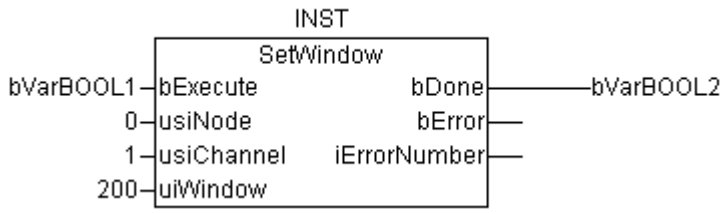
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiWindow := 200)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiWindow := 200);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



14.14 Positioning

AbortMotor

PMCprimo-Command: AB (**Ab**ort, emergency stop)

Function library: **Positioning**

Description:

The motor brakes with the brake ramp SetAbortDeclaration (page 226) up to the velocity 0. This function block can be used with each movement.

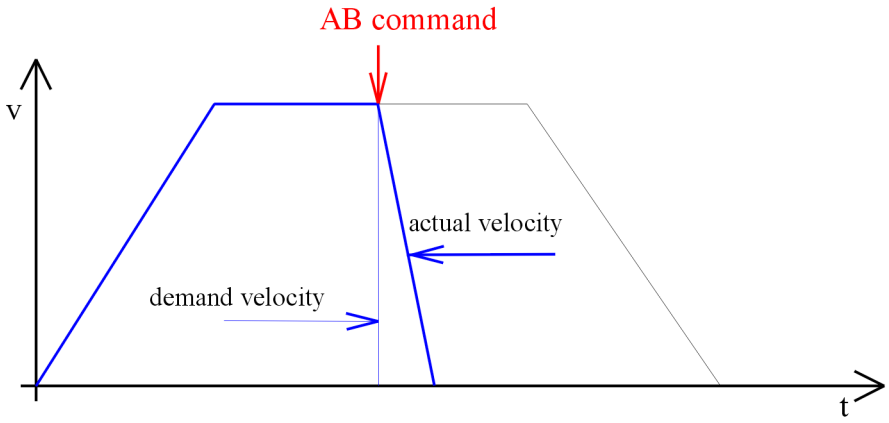


Illustration 20: end of movement with AB-Command

The function block has only an effect on the momentary selected axis.

Input variables:

bExecute (BOOL)	The axis is braked in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Axis has be braked (True) or function block is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected function:

SetAbortDeceleration

Examples:

Declaration:

```
INST: AbortMotor;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

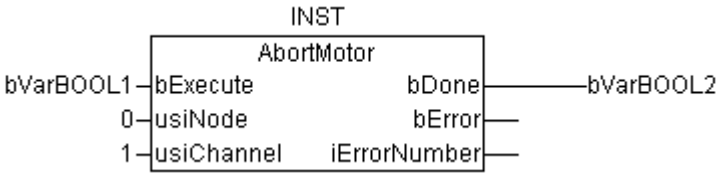
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



GetError

PMCprimo-Command: $Fx.x

Function library: **Positioning**

Description:

This function block returns the actual error value of the axis.

The following error signals can be returned:

Code	Error
0	Internal program error!
1	Program cannot be started, as max. number of running processes is achieved!
2	This command is not implemented on the control!
3	The command MA cannot be executed, as motor off!
4	The target position is outside of the defined software end positions!
5	The software end position in plus-direction has been overrun!
6	The software end position in minus direction has been overrun!
7	The command MA cannot be executed, as motor runs already!
8	The command MR cannot be executed, as motor off!
9	The command MR cannot be executed, as motor runs already!
10	The command VC cannot be executed, as motor off!
11	The command VC cannot be executed, as motor runs already
12	Attempted division by means of zero!
13	Before calling a program you have to switch over to the sequential command sequence!
14	Overflow of the data stack!
15	Motiongenerator runs already
18	Error Motiongenerator: Variable <$MNAME> is not defined.
19	Error Motiongenerator: Variable <$MNPT> is not defined.
20	Variable <...> is already used for <WV>
21	Input is not defined as counting input!
22	Input is not defined as release input!
23	Too less working memory!
24	The following error limit has been exceeded!
25	Input cannot be defined as “reset input”!
26	Trigger variables are not defined!
27	The program <...> cannot be started, as it runs already!

Code	Error
28	The target position could not be achieved!
29	Encoder signals have not been received!
30	Error when reading the encoder
31	The command XM cannot be executed, as motor off!
34	Too much time was required for the setpoint generation!
35	Program <...> cannot be executed, as it is not available!
36	ML must be executed before BR
37	The reference error limit has been exceeded!
38	The reference error could not be corrected in a cycle!
39	The monitoring time for the reference signal has been expired!
40	The reference error has been reduced to the maximum value!
41	Program <...> is momentary not active
42	Velocity allocation only possible with a linear Map
47	The input number must not be bigger than 8!
48	This input module number is not defined!
49	The output number must not be bigger than 8!
50	The output module is not available!
52	The output has already been defined as “error output”!
53	The output has already been defined as “electronic cam”!
54	The output has already been defined for “pulse at cycle limit”!
55	Not defined error message!
56	The limit switch has been activated and motor switched-off!
57	The input is already defined!
58	The input is not defined as “limit switch”!
59	At least 2 value pairs must be defined for the map generation
60	Error Motiongenerator: Variable $MSnnn has not been defined
61	Error Motiongenerator: False value for $MXnn
62	Error Motiongenerator. False value for $MYnn
63	Target position is outside of the software end position
64	Variable for motion generation is not defined
66	The parameter value <...> is not permissible for the command <...> !
67	Command ML cannot be executed, as axis in mapping
68	Error motion generation: variable $MMnn is smaller than the previous value!
69	Input cannot be defined as “input function”!
70	Input cannot be defined as “enable input!
71	Input cannot be defined as “counting input”!

Code	Error
72	Master and slave axis must be different from each other!
73	The axis <...> is not available (Command <...>)!
76	<PC> cannot be executed, as analogue auxiliary output is defined!
77	The output has already been defined as auxiliary output!
78	For this axis an encoder torque consumption has already been defined!
79	Input is not a “fast input”!
80	The “fast input” is already defined
81	A reference signal has not been defined
82	Command IB not possible, as axis virtual and motor off
83	Command LW cannot be executed, as allocation as slave via Slave ML is already made
84	The output is defined and can, therefore, not be manually switched.
85	The output has already been defined as reference output.
86	The output has already been defined as error output for the analogue input limit
87	The output is not defined
88	The output has already been defined as reference error correction
89	For the axis the encoder zero track is already defined
90	For the axis an input is defined as encoder torque consumption
91	For the axis an input is defined as reference input
92	The input is already used for another axis
93	Faster input is not defined as encoder torque consumption
94	Faster input is not defined as reference input
95	The output has already been defined for “pulse issue with a reference signal”
96	Error when incrementing: upper limit of 2*SV has been exceeded
97	Error when incrementing: crawl speed must not get bigger than SV
98	The direction of the velocity must not be changed due to incrementing
99	The command XR cannot be executed, as motor off!
100	The command XR cannot be executed, as motor already runs!
101	Error Motiongenerator: Variable $MFnn has not been defined
102	Error Motiongenerator: Variable $MWnn has not been defined
103	Error Motiongenerator: Variable $MAnn has not been defined
104	Error Motiongenerator: Variable $MZnn has not been defined
105	Error Motiongenerator: Variable $MXnn has not been defined
106	Error Motiongenerator: Variable $MYnn has not been defined
107	Error Motiongenerator: Variable $MXnn > $MYnn

Code	Error
108	Error Motiongenerator: Variable $MBnn has not been defined
109	Error Motiongenerator: Variable $MCnn has not been defined
110	Error Motiongenerator: Segment No.nn: Sum of the percentage indication must result in 100 %
111	Motiongenerator not released
114	The output has already been defined as <JF>
115	Command <XR> or <XM> has been stopped
116	BUS-Variable is not defined
117	Lower limit value for analogue input has been fallen below
118	Upper limit value for analogue input has been exceeded
119	ML must be executed before XV
120	The input is defined for encoder torque consumption
121	The input is defined as reference input
122	Position control loop cannot be activated, as initialisation active
123	Analogue auxiliary output cannot be defined, as position control active
124	It cannot be switched to the virtual mode, as analogue auxiliary output is defined.
125	Transmission ratio can only be incremented, when the position table is active
126	Program <....> is not available
127	False output number
128	False output module number
129	Value range for shift register with Command TC is between1 and 32
130	FC-command only possible with a switched-off position control loop
131	VM-command only possible with a switched-off position control loop
132	It is not possible to couple without software clutch in case of a driving master
133	The command VC cannot be executed, as axis in mapping!
134	Idle loop absolute position is already active on the axis
135	Idle loop relative position is already active on the axis
136	Idle loop reference input is already active on the axis
137	Idle loop cycle limit is already active on the axis
138	Idle loop overflow counter is already active on the axis
139	Axis can only be virtual axis
140	Position table (Map) <...> is not available
141	Master value is not available in a position table (Map)
142	Axis is not available
143	Variable is not defined
144	False supply station number with input description

Code	Error
145	A program is not defined for the input
146	A fast input is already defined for the axis
149	It is not possible to run more than 8 master axes via the CAN-Bus
150	Command UL cannot be executed, as position allocation is active
151	ML must be executed before XM
152	External master value is not allowed with XR
153	Attention data are cancelled, as incorrect check sum calculation
154	Monitoring runs already
155	The following baud rates are adjustable: 9600, 14400, 19200, 38400, 57600, 115200
156	A value for recording has not been selected
157	If monitoring runs, TW must not be modified
158	Channels >4 do not exist
159	False data type TW command
160	Programs are not defined
162	Mode <...> cannot be selected for TC command
163	Incorrect counting value <...> for TC command
165	Attention!!! Error with software update
171	For software differential NL must be executed
172	Controller error: heat sink temperature > 80 °C
173	Controller error: Overvoltage on DC link
174	Controller error: Interruption, short circuit in feedback loop (resolver, Hiperface®)
175	Controller error: Undervoltage on DC link
176	Controller error: Motor limit temperature exceeded
177	Controller error: Internal supply voltage not OK
178	Controller error: Maximum velocity exceeded
179	Controller error: EEprom checksum errorr
180	Controller error: Flash-Eprom checksum error
181	Controller error: Interruption or short circuit with brake
182	Controller error: Motor phase is missing
183	Controller error: Unit internal temperature too high
184	Controller error: Power step defective
185	Controller error: I^2t-limit exceeded
186	Controller error: 2 or 3 phases are missing in the supply voltage
187	Controller error: A/D-Converter defectivet
188	Controller error: Ballast resistor defective or incorrect setting

Code	Error
189	Controller error: Phase is missing in the supply voltage
190	Controller error: System Software is not ok
191	Controller warning: I^2t-step is exceeded
192	Controller warning: Pre-set ballast capacity is achieved
193	Controller warning: Watchdog for extension board (PMCprimo) has responded
194	Controller warning: Phase of the supply voltage is missing
195	Controller warning: Hiperface®: basic setting from motor loaded
196	Controller warning: Extension board (PMCprimo) does not function correctly
197	Controller warning: Enable Signal not switched
198	Error CAN-Bus: Initialisation: Safety lines are not online
199	Error CAN-Bus: When writing PDO-data: CAN not operational in Mode
200	Error CAN-Bus: When writing on the pipe: Buffer full
201	Error CAN-Bus: The safety line has been switched by a participant
202	Error CAN-Bus: The 12V-voltage supply has been switched off
203	Error CAN-Bus: Overflow in the reception buffer. Data went lost
204	Error CAN-Bus: Overflow in the transmission buffer: Data get lost
205	Error CAN-Bus: The CAN-Controller has been switched off due to too many errors
206	Error CAN-Bus: The reception buffer in the CAN-Controller is overflown
207	Error CAN-Bus: An error occurred during transmission
208	Error CAN-Bus: The Watchdog of the CAN-Module has reacted
209	Error CAN-Bus: Errors occurred with the SDO-transmission. Time out
210	Error CAN-Bus: An error has occurred with the SDO-transmission, e.g. number of Bytes wrong
211	Error CAN-Bus: Error message, this message is not implemented
212	Error CAN-Bus: Too many CAN errors: CAN-Bus is switched-off
213	CAN-Bus not active !
214	CAN-nodes nn: Firmware-Version does not fit to Host
215	Error: Servo controller cannot be switched-on via CAN
216	Error: Servo controller cannot be switched-off via CAN
217	PD-command only possible from axis nn!
218	Error: CAN-Bus: Node guarding error, node nn!
219	Attention data loss in a battery back-up memory
220	Battery back-up memory is not available in the system
221	Position table (Map) <...> has not been sent to axis (Command TM)
223	Mapping is not active
224	Error of servo controller via CAN

Code	Error
225	Attention: Bit 1 of FW has been set to 1, as it is RJ>SB
226	Variable could not be allocated to drive command!
227	Data are not available in Flash!
228	Too less Flash-memory !
229	Error has occurred during memorising !
229	Error while storing data !
230	Buffer for storing reference position is full !
231	Direction has changed ! The buffer with reference position must delete
232	Possible baudrates: 9600, 19200 (XOn/XOff), 38400 (XOn/XOff)
233	Output is already defined as 'velocity trigger output
234	To many PLC-tasks active
235	No correct datas for TR
236	No output definition for CAN-IO modul
237	TD %d,%d is not defined!
238	The output is already defined as 'outside window output
239	Cannot execute ID, while axis is not in status 'PC'
240	Parameter <%ld> for command <%c%c> out of range, if bit 4 of RW is set
241	Parameter <%ld> for command <%c%c> out of range, if bit 4 of RW is not set
242	Paramater <%ld> for command <%c%c> out of range, if bit 6 of RW is set
243	Paramater <%ld> for command <%c%c> out of range, if bit 6 of RW is not set
244	Value of RC<1000 not allowed, because bit 7 of RW is set to 0
245	Value of RV>200 not allowed, because bit 5 von RW is set to 0.
246	Bit 5 of RW is set to 1, because bit 7 was set to 1
247	Intern error PLC
248	Cannot execute drive command, because node is no drive
249	Drive error: Commutation error
250	Variable $%s[%d] is not defined
251	Node number not defined (command <%c%c)
252	Node number %d is already in use (command <%c%c)
253	No CAN-device found
254	Cannot execute drive command, because buffer overflow
255	Warning: SRam battery low
256	Drive error: Enable switched on before AS-option
257	Drive error: Earth short circuit
258	Drive warning from CAN
259	%d bytes Compact Flash memory free

Code	Error
260	Cannot execute <XM> while motor is running
261	Command QA0:CAN-Adr can only done on the host
262	The analog output <%ld> is not available! (command <%c%c>)
263	Reserved for Soft-SPS
264	Reserved for Soft-SPS
265	Reserved for Soft-SPS
266	Reserved for Soft-SPS
267	Reserved for Soft-SPS
268	Reserved for Soft-SPS
269	Reserved for Soft-SPS
270	Reserved for Soft-SPS
271	Reserved for Soft-SPS
272	Reserved for Soft-SPS
273	PD command not possible! PMCtendo DD4 version wrong(< 4.94)
274	Node %d, Drive command not completed !
276	Battery buffered memory is used by SoftPLC
277	The maximum possible position (SB*SM) is greater than the position limit of %ld (command <%c%c>)
278	The predefined map 'LINEAR' can't be changed

Input variables:

bExecute (BOOL)	The error value is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been read (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	Error message according to Table

See also:

GetStatus

Examples:

Declaration:

```
INST: GetError;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

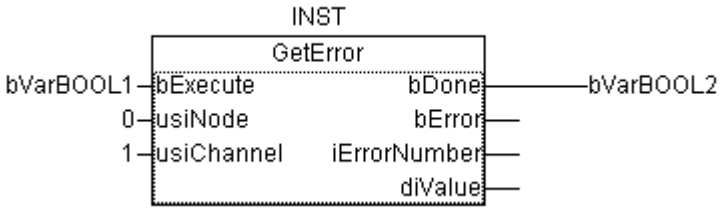
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.biValue
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



GetStatus

PMCprimo-Command: $Sx.x

Function library: **Positioning**

Description:

This function block restores the actual status of an axis.

Summary of all Codes which PMCprimo restores depending on the status of the axis.

0 Position control loop closed (EnablePositionControl)

8 Position control loop open, motor off (MotorOff)

66 Axis waits on the coupling at the beginning of a position allocation out of (C)

67 Axis executes a coupling process at the beginning of a position allocation out of (C).

68 Axis executes a compensating movement for the beginning of a position allocation (A)

256 Execution of an endless positioning or velocity control (MoveConstantVelocity)

512 Axis executes a Positioning Command (MoveToAbsolutePosition, MoveRelativePosition)

2048 Positioning allocation (= Map) active (ExecuteMap)

4096 Axis executes a Stop-command (StopMotor)

8192 Initialisation runs (InitialisePosition, InitialisePositionBounds)

Input variables:

bExecute (BOOL)	The status is read in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0 except diValue
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Value has been read (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)
diValue (DINT)	Error message according to Table

See also:

GetError

Examples:

Declaration:

```
INST: GetStatus;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
diValue:DINT;
```

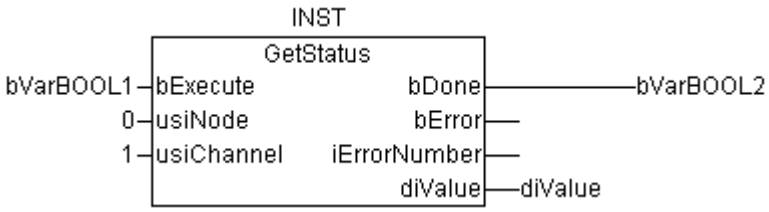
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
LD    INST.biValue
ST    diValue
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1);
bVarBOOL2:=Inst.bDone;
diValue:=Inst.diValue;
```

Example in FBD:



InitialisePosition

PMCprimo-Command: IN (Initialise position)

Function library: **Positioning**

Description:

PMCprimo executes an initialisation function so long, until a reference signal is recognised. During the initialisation PMCprimo signals „ I “.

Execution: The motor accelerates to the demand velocity and keeps the same so long constant, until PMCprimo detects a reference signal. The direction, to which the motor drives, is specified via an argument. The position counter is immediately set to the value of SetReferenceOffset (Reference offset page 271) and the motor is stopped. Afterwards the motor moves back to the position 0.

The function block for the initialisation can also be used with an open position control loop. In this case PMCprimo waits for a reference signal and when detecting a reference signal the position counter is set to the value of SetReferenceOffset.

The return to the reference signal can be prevented with the function block SetReferenceOptionsWord (page 272).

The function block SetReferenceTimeout (page 280) also acts with the execution of this function block.

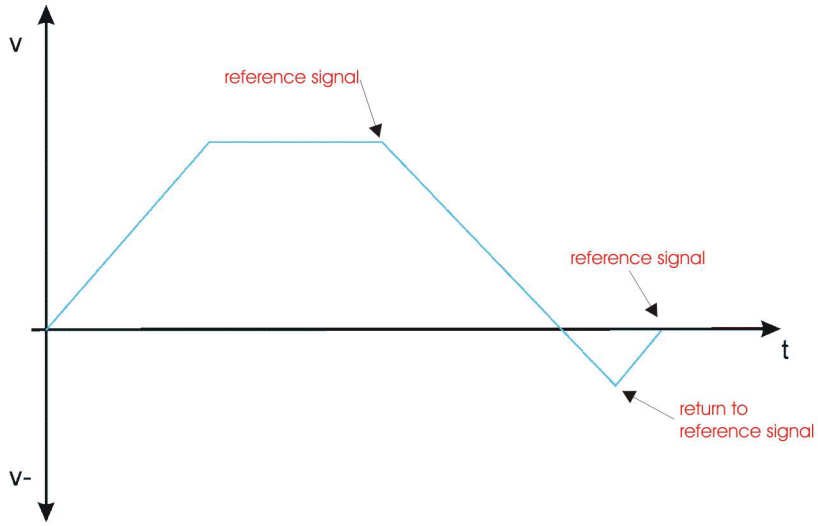


Illustration 21: Initialisation with return to the reference signal

Input variables:

bExecute (BOOL)	The initialisation is started in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
bDirection (BOOL)	The rotational direction 0: negative 1: positive

Output variables:

bDone (BOOL)	Initialistion completed (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

See also:

InitialisePositionBounds

Examples:

Declaration:

```
INST: InitialisePosition;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

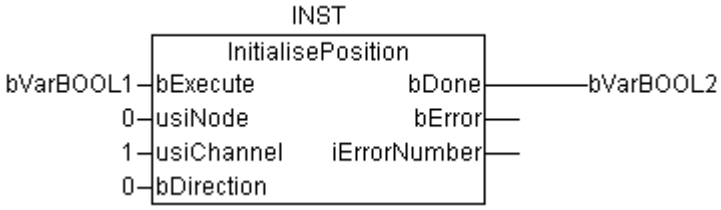
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, bDirection := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, bDirection := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



InitialisePositionBounds

PMCprimo-Command: IB (Initialise position and bounds)

Function library: **Positioning**

Description:

This function block is similar to the function block InitialisePosition. With this function block the position bound of the machine is also determined additionally to the initialisation of the position. PMCprimo executes an initialisation command at the selected axis so long, until a reference signal is recognised for the first time. Following the axis drives further with constant velocity, until the reference signal is recognised for the second time. PMCprimo considers the distance between recognising the reference signal as position bound of the selected axis. PMCprimo signals “I “during the initialisation.

Execution: The motor accelerates to the setpoint velocity and keeps the same so long constant, until PMCprimo recognises a reference signal. The direction, to which the motor drives, is determined via an argument. The position counter is immediately set to the value of SetReferenceOffset (Referenzoffset page 271) and the motor is braked. Following the motor drives back to the position 0.

The function block for the initialisation can also be used with an open position control loop. In this case PMCprimo waits for a reference signal and when recognising a reference signal the position counter is set to the value of SetReferenceOffset. Following the distance between reference signals as position bound is entered to the value SetPositionBound.

The return to the position 0 can be prevented with the function block SetReferenceOptionsWord (page 272).

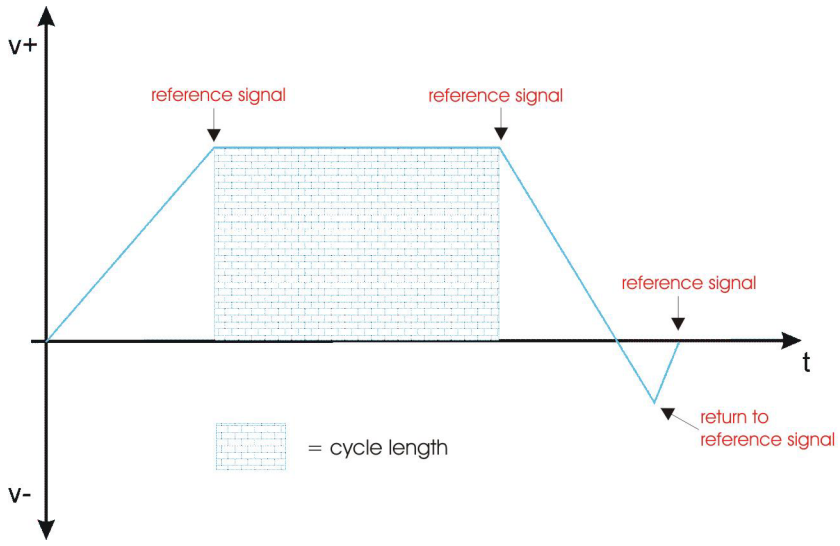


Illustration 22: Initialisation position and position bound

Input variables:

bExecute (BOOL)	The initialisation is started in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
bDirection (BOOL)	The rotational direction 0: negative 1: positive

Output variables:

bDone (BOOL)	Initialistion completed (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

See also:

InitialisePosition

Examples:

Declaration:

```
INST: InitialisePositionBounds;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

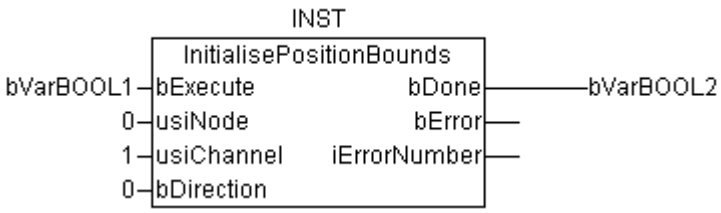
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, bDirection := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, bDirection := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



MoveConstantVelocity

PMCprimo-Command: VC (Move at **c**onstant **v**elocity)

Function library: **Positioning**

Description:

This function block is used, to drive the motor with a constant velocity in the specified direction. PMCprimo accelerates the motor with the acceleration value SetAcceleration (page 227), until it has achieved the velocity determined with the function block SetVelocity (page 240). The motor drives with constant velocity until to a stop signal is detected.

PMCprimo signals the command execution with the sign “V” at the serial interface. Move commands can only be executed with a closed position control loop.

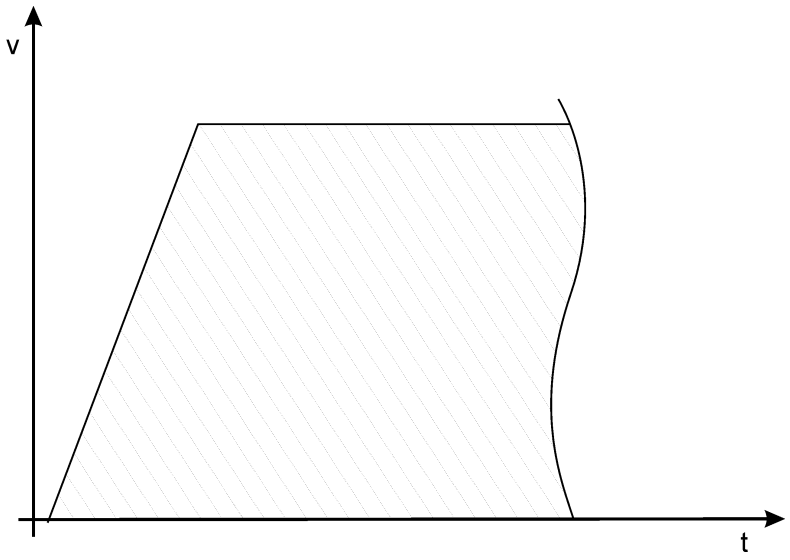


Illustration 23: Move with constant velocity

Input variables:

bExecute (BOOL)	The movement is started in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axle number from 1 to n (depending on the system)
bDirection (BOOL)	The rotational direction 0: negative 1: positive

Output variables:

bDone (BOOL)	Movement started (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

See also:

SetAcceleration, SetVelocity, SetSlowSpeedMode and SetSlowSpeed

Examples:

Declaration:

```
INST: MoveConstantVelocity;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

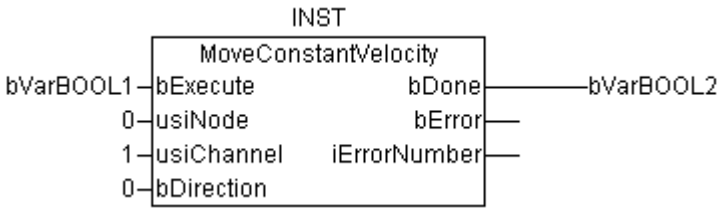
Example in IL:

```
CAL     INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, bDirection := 0)
LD INST.bDone
ST bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, bDirection := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



MoveRelativePosition

PMCprimo-Command: MR (**M**ove units **r**elative to current position)

Function library: **Positioning**

Description:

The motor drives the specified relative position. The motor moves from its instantaneous position by the set increments. The movement follows to trapezoidal or sinusoidal velocity profiles (Illustration 24).

The motor accelerates and stops with the acceleration values which have been specified with the function blocks SetAcceleration and SetDeceleration. The travelling speed is defined with function block SetVelocity. The position is indicated in increments. When using MoveRelativePosition für rotary axes see also function block SetPositionBound.

PMCprimo signals the command execution with the sign “M”. Travel commands can only be executed with an active position control loop. A relative positioning is independent on the Bits 1 – 3 of the function block SetMoveOptionsWord.

During an execution of this function the value of the velocity, set with the function blocks SetVelocity or SetSlowSpeed, can be modified. The switchover of the velocity step can take place with the module SetSlowSpeedMode at any time, however, the value for SetSlowSpeed must always be smaller than the value of SetVelocity.

The target position is checked before the execution of a travelling command as to the observance of the software limits (function blocks SetLowPositionLimit and SetHighPositionLimit). In case the target position is outside of the software limits, The travelling command is not executed and PMCprimo signals the error “target position is outside of the software limits”.

It is possible to leave a position allocation (see function block ExecuteMap page 92) with the function MoveRelativePosition.

Input variables:

bExecute (BOOL)	The movement is started in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diValue (DINT)	The relative target position in increments in the range of ±8.389.000

Output variables:

bDone (BOOL)	Target position has been achieved (True) or movement is still not completed (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetAcceleration, SetVelocity, SetDeceleration, SetSlowSpeedMode, SetLowPositionLimit, SetHighPositionLimit and SetSlowSpeed

Examples:

Declaration:

```
INST: MoveRelativePosition;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

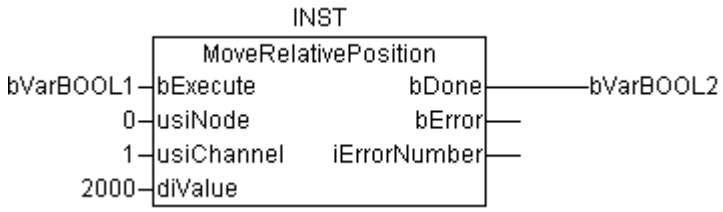
Example in IL:

```
CAL     INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, diValue := 2000)
LD INST.bDone
ST bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, diValue := 2000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



MoveToAbsolutePosition

PMCprimo-Command: MA (Move to absolute position)

Function library: **Positioning**

Description:

The motor moves to the specified absolute position. The movement follows a trapezoidal or sinusoidal velocity profile (Illustration 24). It can be selected via Bit 2, the function block SetControlWord, if it is accelerated trapezoidally or sinusoidally. The motor accelerates with the accelerate value of SetAcceleration (page 227) and stops with the ramp of SetDeceleration (page232). The travel velocity is defined with the function block SetVelocity (page 240). The position is indicated in increments. When using MoveToAbsolutePosition for cyclic axes see also function block SetPositionBound (page 133).

It is possible to complete a position allocation (function block ExecuteMap) with this module. It can be determined via Bit 6 of SetMapOptionsWord (page 127), if it is moved with the actual or with the velocity defined by SetVelocity to the specified position. If Bit 2 of SetMoveOptionsWord is not set, the instantaneous cycle is not left during uncoupling. If necessary, it is stopped and travelled to the demand position in an opposite direction. If Bit 2 of SetMoveOptionsWord (see page 234) is set, the direction defined with Bit 3 is always kept. If this does not coincide with the instantaneous direction, it is modified.

If Bit 1 of SetMoveOptionsWord is set, the shortest way is always travelled.

Enhancement as of **version 1.008a**:
While clutching out of a map with MoveToAbsolutePosition, the actual velocity is compared with SetSlowSpeed. If the velocity of the slave is lower resp. equal SetSlowSpeed, than SetVelocity is used.

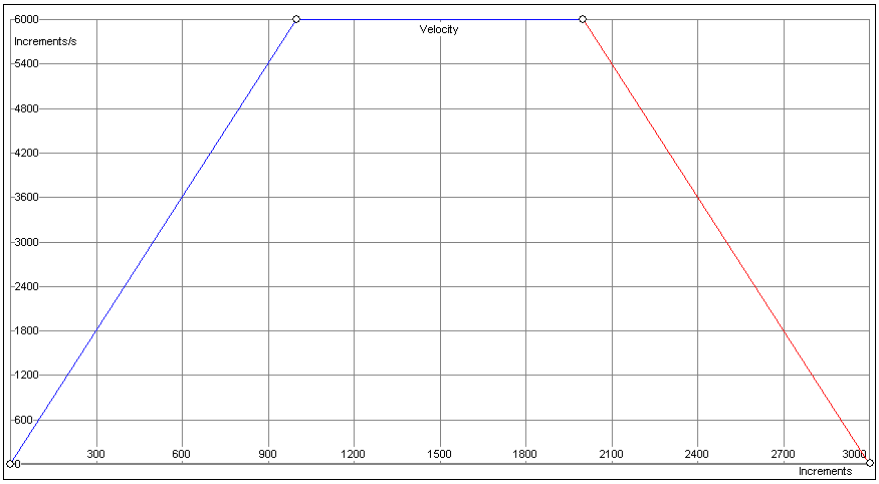


Illustration 24: Trapezoidal velocity profile

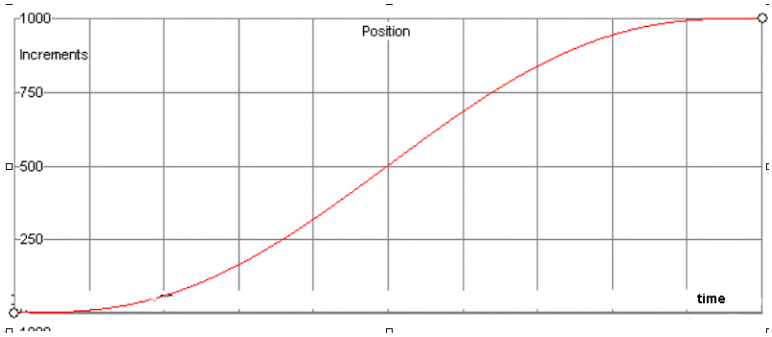


Illustration 25: Positions course with trapezoidal velocity profile

The required velocity cannot be achieved with a very low acceleration value which results in a triangle instead of a trapezoidal velocity profile.

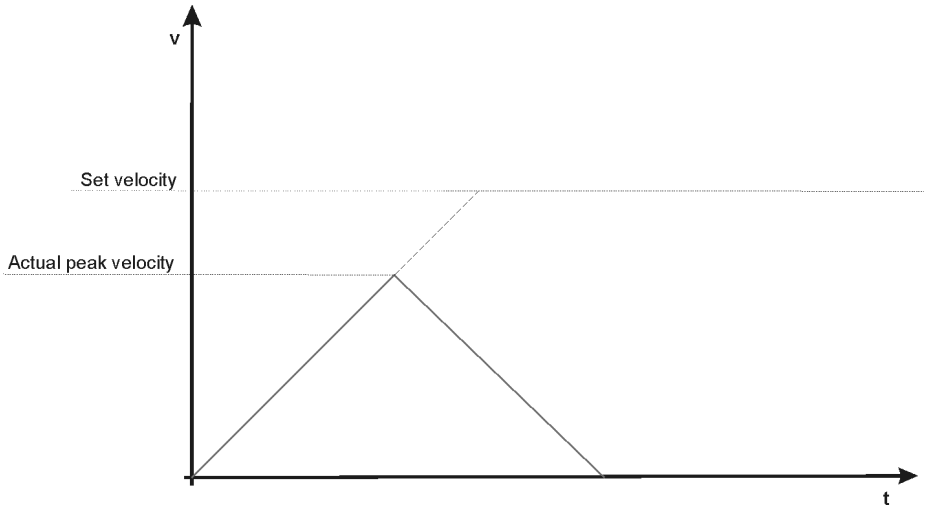


Illustration 26: Triangle velocity profile

PMCprimo signals the command execution with sign “M” at the serial interface. Travel commands can only be executed with an active position control loop.

The target position is checked before execution of a move command as to the observance of the software limit (function blocks SetLowPositionLimit and SetHighPositionLimit). If the target position is outside of the software limits, the move command is not executed and PMCprimo signals the error “target position is outside of the software limits”.

Input variables:

bExecute (BOOL)	The movement is started in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diValue (DINT)	The absolute target position in increments in the range of ±4.000.000

Output variables:

bDone (BOOL)	Target position has been achieved (True) or movement is still not completed (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetAcceleration, SetVelocity, SetDeceleration, SetSlowSpeedMode, SetLowPositionLimit, SetHighPositionLimit, SetMoveOptionsWord and SetSlowSpeed

Examples:

Declaration:

```
INST: MoveToAbsolutePosition;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

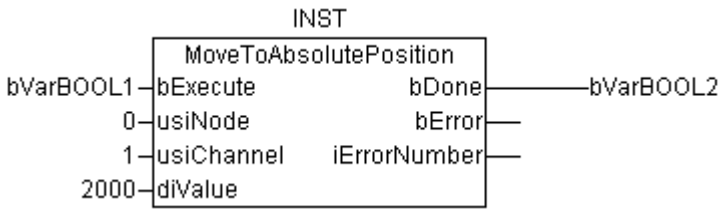
Example in IL:

```
CAL     INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, diValue := 2000)
LD INST.bDone
ST bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, diValue := 2000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



ResetError

PMCprimo-Command: $Fx.x=0

Function library: **Positioning**

As of **version 1.010** available

Description:

With this function block a channel error is erased and also the error message at the LCD display is cleared.

Input variables:

bExecute (BOOL)	The error is cleared in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
udiValue (UDINT)	The brake ramp of 1000 to 2.000.000.000 increments/second²

Output variables:

bDone (BOOL)	Error was cleared (True) or it is still under operation(False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Examples:

Declaration:

```
INST: ResetError;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

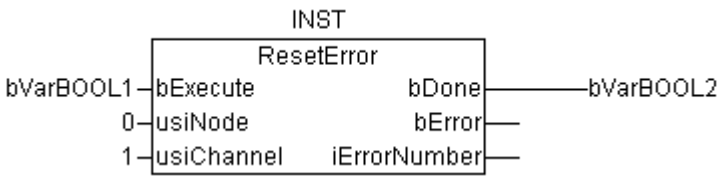
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetAbortDecelaration

PMCprimo-Command: XA (Set deceleration for AB command)

Function library: **Positioning**

Description:

This function block is used to specify the brake ramp for the abort command in increments/second^2. The deceleration ramp can be modified at any time. The smallest adjustable ramp is 1000. This deceleration ramp is only used with the abort function AbortMotor (page 201).

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
udiValue (UDINT)	The brake ramp of 1000 to 2.000.000.000 increments/second²

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation(False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

AbortMotor

Factory setting: 1.000.000 increments/second²

Examples:

Declaration:

```
INST: SetAbortDeceleration;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

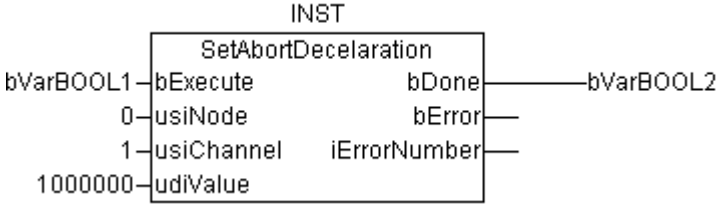
Example in IL:

```
CAL     INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, udiValue := 1000000)
LD INST.bDone
ST bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, udiValue := 1000000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetAcceleration

PMCprimo-Command: SA (Set acceleration)

Function library: **Positioning**

Description:

This function block is used, to specify the acceleration value in increments/second². The acceleration value can be modified at any time. The lowest possible acceleration value is 1000.

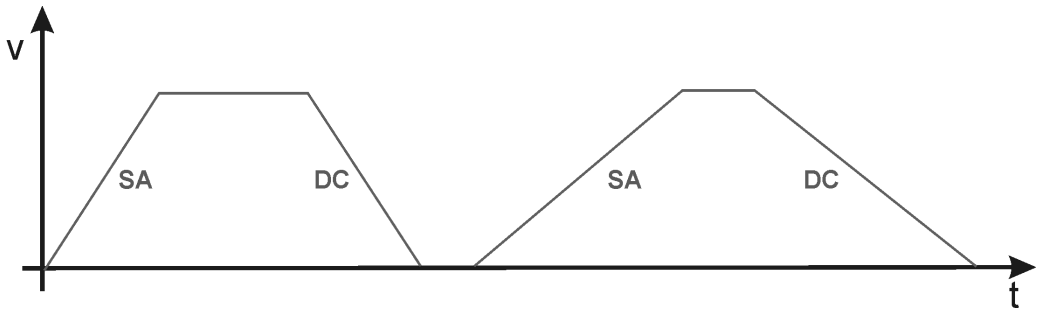


Illustration 27: Higher and low acceleration value

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
udiValue (UDINT)	The value of 1000 to 2.000.000.000 increments/second²

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation(False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

MoveToAbsolutePosition, MoveRelativePosition and MoveConstantVelocity

Factory setting: 100.000 increments/second²

Examples:

Declaration:

```
INST: SetAcceleration;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

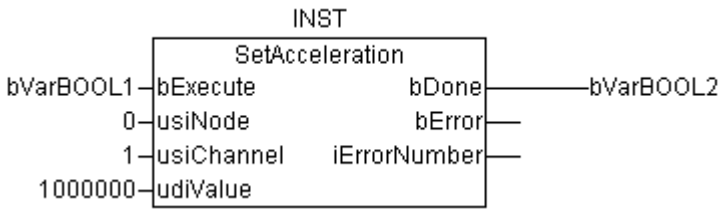
Example in IL:

```
CAL         INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, udiValue := 1000000)
LD INST.bDone
ST bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, udiValue := 1000000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetBacklashDistance

PMCprimo-Command: BL (Set backlash compensation distance)

Function library: **Positioning**

Description:

This function block allows the compensation of a mechanical backlash. After every change of the direction of the motor, the distance, defined in increments, is added to the actual position. This backlash compensation acts only with the execution of MoveToAbsolutePosition or MoveRelativePosition.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
udiValue (UDINT)	The compensation from 0 to 65.535 increments

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation(False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

MoveToAbsolutePosition, MoveRelativePosition

Factory setting: 0 (switched-off)

Examples:

Declaration:

```
INST: SetBacklashDistance;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

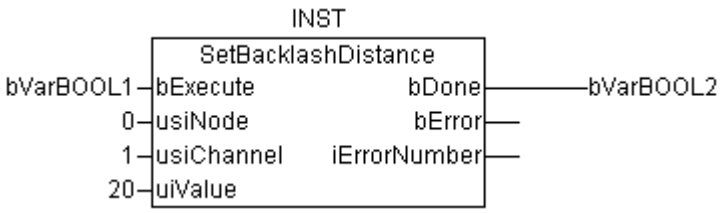
Example in IL:

```
CAL     INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiValue := 20)
LD      INST.bDone
ST      bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue := 20);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetCreepDistance

PMCprimo-Command: SC (**S**et **c**reep distance)

Function library: **Positioning**

Description:

The standard trapezoidal velocity profile of a positioning movement can be completed by the introduction of a way with creep speed for the positioning. The creep speed is set with the function block SetSlowSpeed. The creep speed is used for a slow move of the target position. This command is only effective when SetSlowSpeedMode (page 238) is set to 0.

The deceleration action is initiated so early that the axis travels the slow creep speed 200 increments the latest, before achieving the target position.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiValue (UINT)	The way with creep speed from 0 to 65.535 increments

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation(False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

MoveToAbsolutePosition, MoveRelativePosition, SetSlowSpeed und SetSlowSpeedMode

Factory setting: 0 (switched-off)

Examples:

Example in IL:

Declaration:

```
INST: SetCreepDistance;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

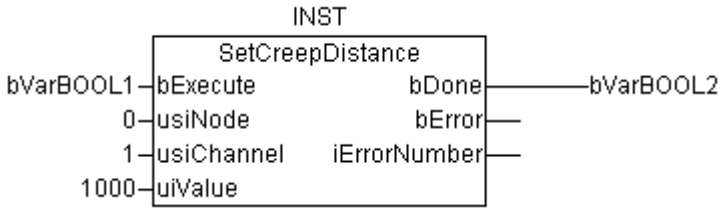
Example in IL:

```
CAL     INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiValue := 1000)
LD INST.bDone
ST bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue := 1000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetDeceleration

PMCprimo-Command: DC (Set deceleration for ST command)

Function library: **Positioning**

Description:

This function block is used, to specify the deceleration ramp in increments/second^2. The deceleration ramp can be changed at any time. The smallest adjustable deceleration ramp is 1000. The deceleration ramp is used with the function blocks StopMotor, MoveToAbsolutePosition, MoveRelativePosition, InitialisePosition and InitialisePositionBounds. A separate deceleration ramp can be set with SetAbortDecelaration for the abort command AbortMotor.

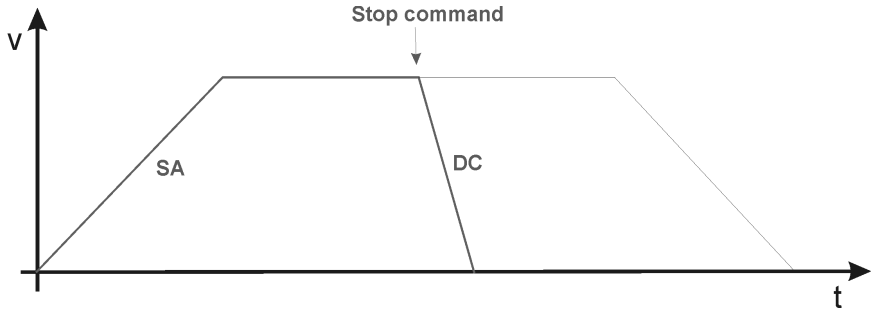


Illustration 28: Set brake ramp

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
udiValue (UDINT)	The value from 1000 up to 2.000.000.000 increments/second²

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation(False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 166)

Connected functions:

StopMotor, MoveToAbsolutePosition, MoveRelativePosition, InitialisePosition and InitialisePositionBounds

Factory setting: 100.000 increments/second²

Examples:

Example in IL:

Declaration:

```
INST: SetDeceleration;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

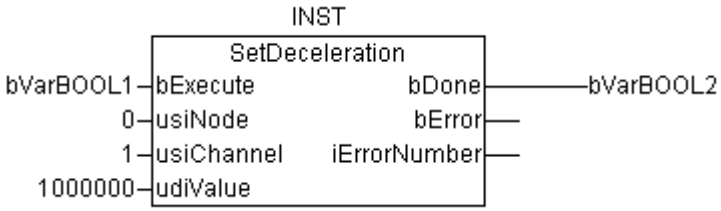
Example in IL:

```
CAL     INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, udiValue := 1000000)
LD INST.bDone
ST bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, udiValue := 1000000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetMoveOptionsWord

PMCprimo-Command: ZW (Set move position control word)

Function library: **Positioning**

Description:

With this function block the user can determine the behaviour of PMCprimo with positioning commands (In case of Input and Output Bit 0 stands on the right and Bit 7 on the left side).

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiControlword (USINT)	Each Bit has the following meaning:

Bit 0: reserved

Bit 1: Determines, if the position bound has an influence on the positioning commands.

0: The position bound set with SetPositionBoundn has an influence on the execution of positioning commands.

1: The position bound is considered with the execution of an absolute positioning (MoveToAbsolutePosition). In case of an absolute positioning the axis travels to the position within the position bound which corresponds to the absolute position (Example: rotary axis travels not 1,5 but 0,5 machine cycles).

Bit 2: This Bit determines that the starting direction of the absolute setpoint positions is determined at the rotary axes with Bit 3.

0: Direction for starting of the setpoint position not defined.

1: Direction for starting of the setpoint position can be defined with Bit 3.

Bit 3: This Bit determines the starting direction of the setpoint position with an absolute positioning at a rotary axis, if Bit 2 is set to 1.

0: The positioning is executed in a positive starting direction.

1: The positioning is executed in a negative starting direction.

Bit 4: **Version 2.000 or higher**:
If bit 1 of the option word is set and the target position is a multiple of the value of SetPositionBound for a move absolute this bit decides to move one bound or not.

0: If bit 1 is set and driveway=n*position bound: no movement.

1: If bit 1 is set and driveway=n * position bound: move one bound.

Bit 5: reserved

Bit 6: reserved

Bit 7: **Version 2.000 or higher:**

If the axis is not moving in mapping an automatic bound correction is possible with this bit. Therefore the position bound is multiplied with the scale map set by the function block SetScaleMapping. With an odd gear transmission a reference sensor is no longer necessary.
Example: position bound=4096; scale map= 1:3
The bound for it is 1365,33. Therefore a drift of one increment every 3 bound would happened. With correction the bound set two cycles to 1365 and one cycle to 1366. The calculated bound can be shown with the function GetMappedMasterBound.

0: Automatic correction is not active
1: Automatic correction is active

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation(False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Factory setting: 0

Examples:

Example in IL:

Declaration:

```
INST: SetMoveOptionsWord;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

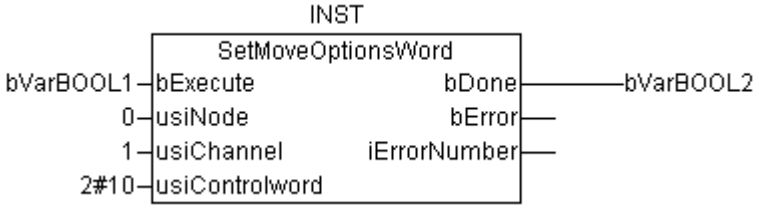
Example in IL:

```
CAL     INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, usiControlword := 2#10)
LD INST.bDone
ST bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiControlword := 2#10);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetSlowSpeed

PMCprimo-Command: SS (**S**et **s**low speed)

Function library: **Positioning**

Description:

With this function block the velocity of the creep speed is specified in increments/second. The velocity of the creep speed is the travel velocity of the axis, when SetSlowSpeedMode (page 238) is set to 1. The value for the execution of positioning command must be smaller than the value SetVelocity (page 240).

The velocity SetSlowSpeed is also used with a reference equalising travel when the axis should stand.

Additionally it is also used for an adjustment of SetMapBaseOffset, SetSlaveMapOffset and SetScaleMapping with a stopped master.

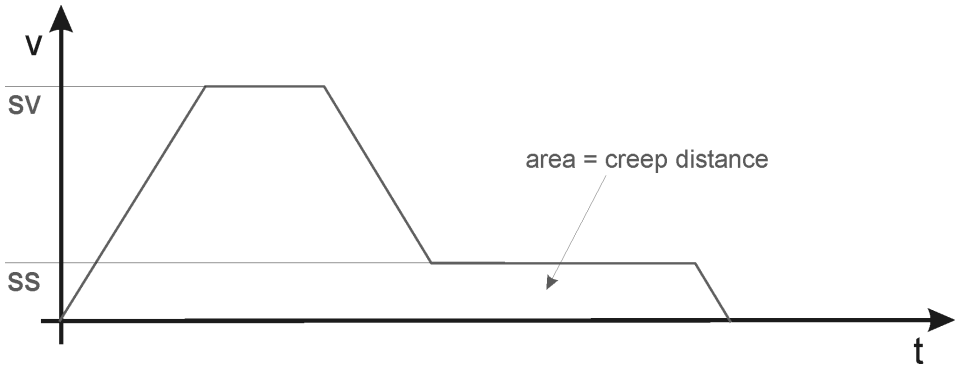


Illustration 29: positioning with creep speed

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
udiValue (UDINT)	The velocity from 0 to 4.000.000 increments/s

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation(False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetSlowSpeedMode, MoveConstantVelocity, MoveRelativePosition, MoveToAbsolutePosition, SetReferenceCorrectionVelocity, SetMapNBaseOffset, SetSlaveMapOffset and SetScaleMapping

Factory setting: 0 (stoppage)

Examples:

Example in IL:

Declaration:

```
INST: SetSlowSpeed;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

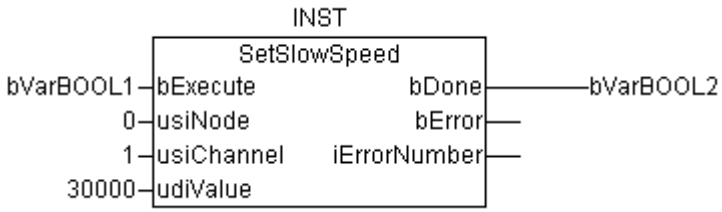
Example in IL:

```
CAL     INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, udiValue := 30000)
LD INST.bDone
ST bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, udiValue := 30000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetSlowSpeedMode

PMCprimo-Command: VJ (Set slow velocity mode)

Function library: **Positioning**

Description:

With this function block it is switched over between the velocity values SetVelocity and SetSlowSpeed. With SetSlowSpeedMode = 0 the axis moves with SetVelocity (page 240). With SetSlowSpeedMode = 1 the axis moves the velocity set with SetSlowSpeed. The adjustment of a creep speed (function block SetCreepDistance page 230) is only valid with SetSlowSpeedMode = 0.

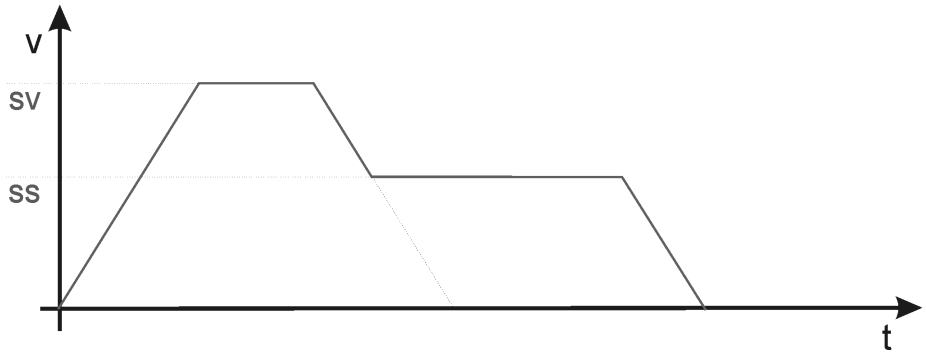


Illustration 30: Velocity steps SetVelocity, SetSlowSpeed

In case of a switchover between SetVelocity and SetSlowSpeed during the execution of MoveToAbsolutePosition or MoveRelativePosition the specified value for SetSlowSpeed must always be smaller than the value for SetVelocity, as otherwise the deceleration ramp cannot be calculated correctly (deceleration ramp very steep).

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiValue (USINT)	The velocity step (0 or1)

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation(False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetSlowSpeedMode, MoveConstantVelocity, MoveRelativePosition, MoveToAbsolutePosition, SetReferenceCorrectionVelocity, SetMapBaseOffset, SetSlaveMapOffset and SetScaleMapping

Factory setting: 0 (standard velocity step)

Examples:

Example in IL:

Declaration:

```
INST: SetSlowSpeedMode;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

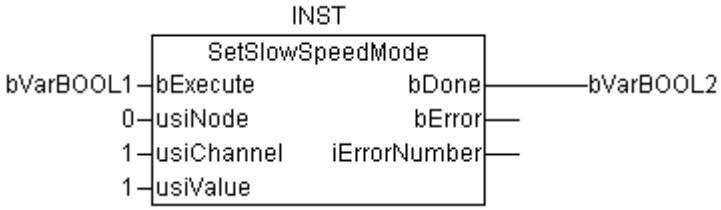
Example in IL:

```
CAL     INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, usiValue := 1)
LD INST.bDone
ST bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiValue := 1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetVelocity

PMCprimo-Command: SV (Set velocity)

Function library: **Positioning**

Description:

With this function block the velocity is specified in increments/second. The velocity can also be changed during move

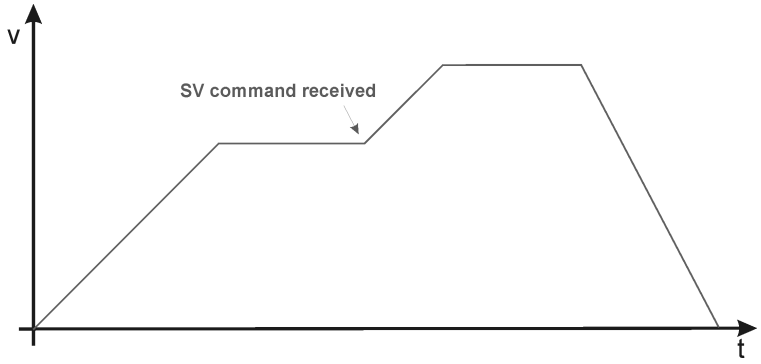


Illustration 31: Change travel velocity during a move

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
udiValue (UDINT)	The velocity from 0 to 4.000.000 increments/s

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetSlowSpeedMode, MoveConstantVelocity, MoveRelativePosition and MoveToAbsolutePosition

Factory setting: 20.000 Increments/s

Examples:

Example in IL:

Declaration:

```
INST: SetVelocity;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

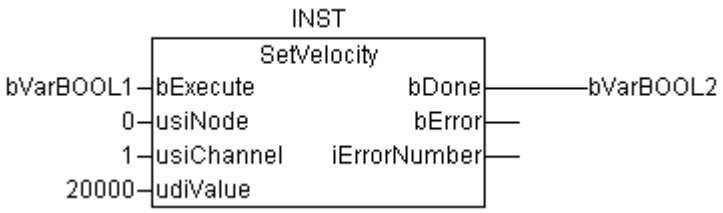
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, udiValue := 20000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, udiValue := 20000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



StopMotor

PMCprimo-Command: ST (**St**op)

Function library: **Positioning**

Description:

The motor decelerates the specified ramp down to the velocity 0 with the function block SetDeceleration (page 232). This function block can be used with each movement. PMCprimo signals “S" during the deceleration process.

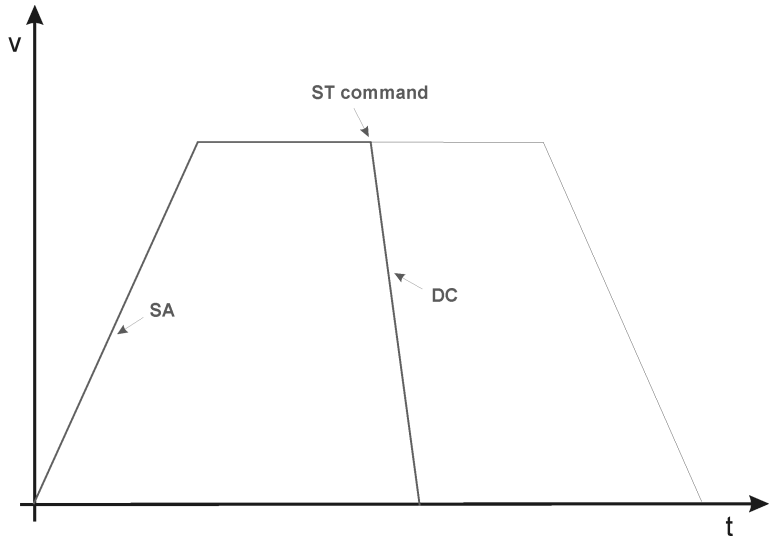


Illustration 32: End of movement with ST-Command

The function StopMotor acts only on the instantaneous selected axis. The function block GlobalStop (page 178) acts axis overlapping.

Input variables:

bExecute (BOOL)	The axis is stopped in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	Axis has been stopped (True) or axis is braking at the moment (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

MoveConstantVelocity, MoveRelativePosition, ExecuteMap, InitialisePosition, InitialisePositionBounds and MoveToAbsolutePosition

Examples:

Example in IL:

Declaration:

```
INST: StopMotor;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

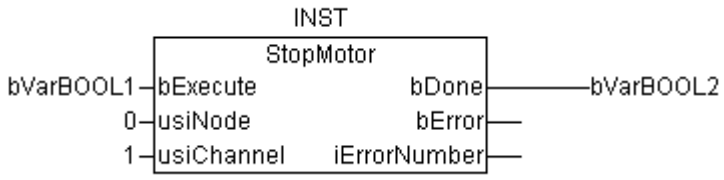
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



StopMotorToPosition

PMCprimo-Command: STnn (**St**op)

Function library: **Positioning**

Description:

The motor stops at the specified position with the instantaneours velocity and the deceleration ramp specified with the function block SetDeceleration (page 232). The command SetMotorToPosition can be used with every movement. PMCprimo signals “S” during this process. The target position must be ≤ SetPositionBound value.

Enhancement as of **version 2.004**:

In the past the command stop to position (example “**diPosition = 0**”) with mapping the active map was only kept until the deceleration was started. Then a linear ramp (ramp depending from “**SetClutchTime**” or “**SetClutchLength**”) was used to stop the slave. If the master was stopped in that time then the slave still moved to the target position because the master slave link was already opened. The slave moved therefore sometimes more than the master.

Now with “**SetMapOptionsWord**” bit 6 it is possible to maintain the master slave link also while decelerating until the final target position is reached. This means that the slave is stopping as the master and then waiting until the master is moving again until the slave reaches the position. The setting of “**CL**” is always used. This means the bit 5 of “**SetMapOptionsWord**” (SetClutchTime / SetClutchLength setting) (see page 127) is ignored because with command “**SetClutchTime**” it is not possible to reach the target position.

Input variables:

bExecute (BOOL)	The axis is stopped in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diPosition (DINT)	The absolute target position in increments in the range of ±4.000.000 Version **2.004** new range: ±2.000.000.000

Output variables:

bDone (BOOL)	Axis has been stopped (True) or axis is braking at the moment (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

MoveConstantVelocity, MoveRelativePosition, ExecuteMap, InitialisePosition, InitialisePositionBounds and MoveToAbsolutePosition

Examples:

Example in IL:

Declaration:

```
INST: StopMotorToPosition;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

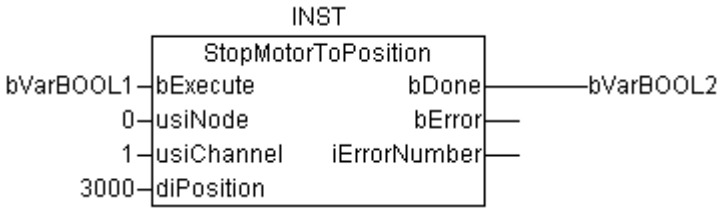
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, diPosition := 3000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, diPosition := 3000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



14.15 Referencing

Activation and adjustment of the reference behaviour.

This chapter describes the commands and, therefore, the possibilities, which PMCprimo offers the user regarding the referencing. Reference function blocks include the basic function of the initialisation after power on as it is integrated in every positioning system, but also the possibility of the cyclic referencing, which allows the user to adjust fluctuations and inaccuracies of the practical position bound to the theoretical specification of the position bound.

The encoder position is stored immediately when recognising a reference input. This stored position is compared with an expected reference position (cycle limit: -SBn / 0 / +SBn; see function block SetPositionBound). The resulting difference is defined as reference error and the absolute position can, if required, be corrected by this difference amount.

PMCprimo supports 2 kinds of reference signals:

A: The zero track of the encoder is connected with a fast reference input, which recognises itself pulses with a length of 60ns. This reaction time is short enough, in order to recognise itself the zero track in case of high-resolution encoders with their maximum velocity (definition >zero track is reference input< with function DefineZeroMarkerInput).

B: Digital inputs as reference inputs. The inputs 1 to 4 (or 1 to 2 with ‘PMCprimo Drive‘) can be defined with the function block DefineReferenceInput as reference inputs. With this reference inputs external switching elements can be used with a minimum pulse length of 10 µs for the referencing.

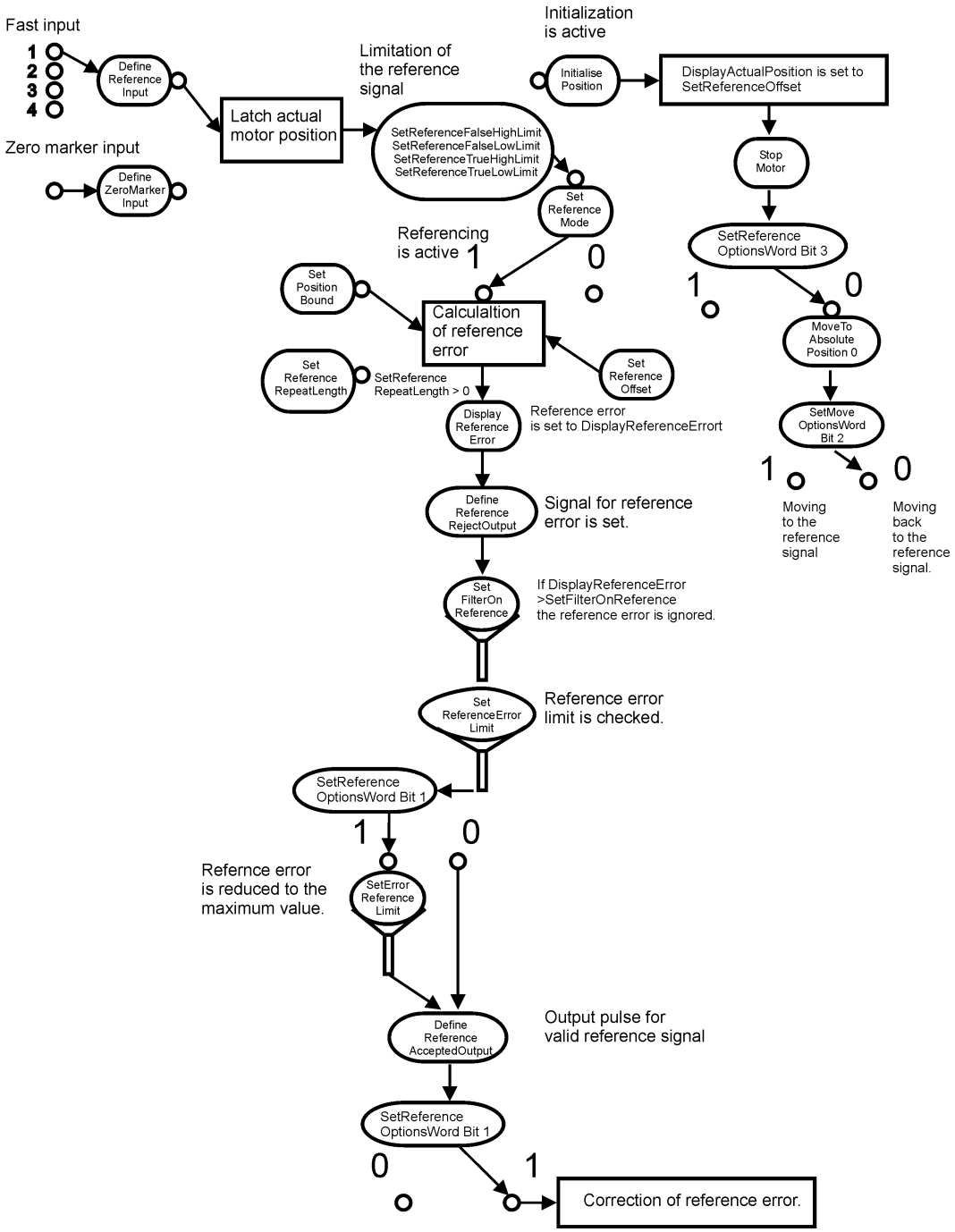


Illustration 33: Cooperation of the commands after detecting a reference signal.

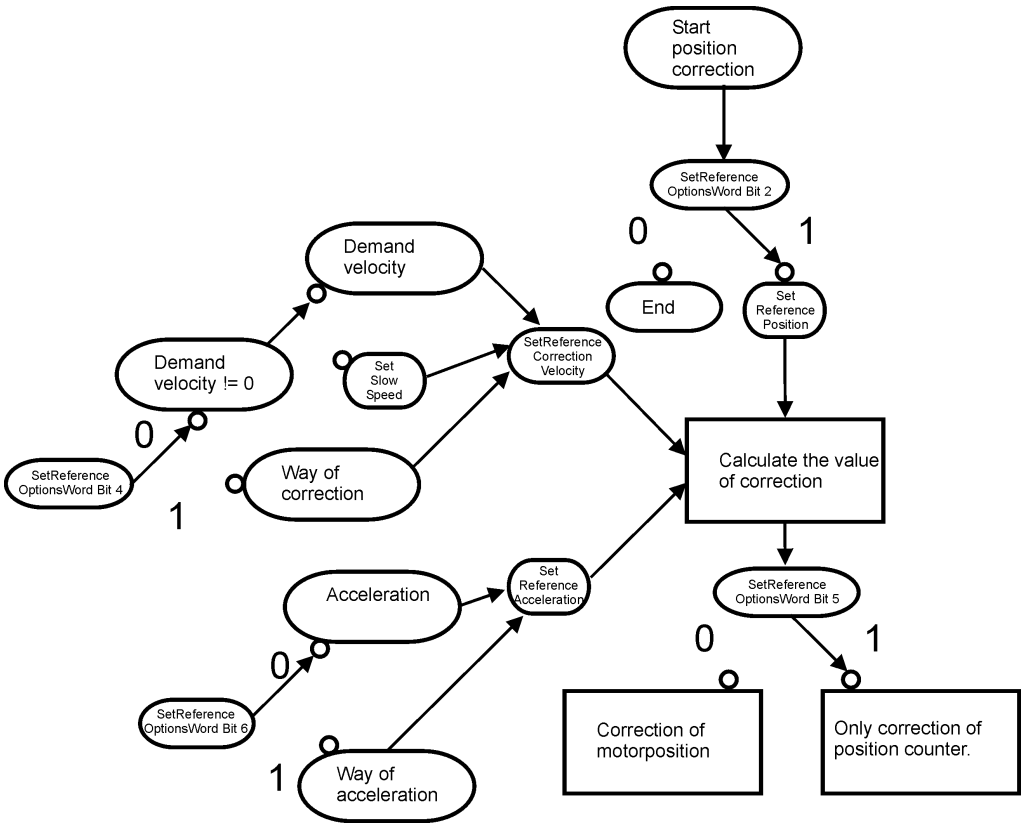


Figure 34: Cooperation of the commands reference error correction

DefinePositionSnapshot

PMCprimo-Command: PS (Define **p**osition **s**napshot input)

Function library: **Referencing**

Description:

With this function block an encoder position snapshot can be stored with a digital input. The stored value can be read with the function block DisplaySnapshotPosition (see page 63). Only fast inputs for the definition can be used. The encoder position snapshot is only executed when the reference function is activated with SetReferenceMode. The stored value of the encoder position snapshot is independent from a reference offset set with SetReferenceOffset.

It is not possible to define additionally a position snapshot at this axis if a reference signal (function block DefineReferenceInput or DefineZeroMarkerInput) is already been defined.

Input variables:

bExecute (BOOL)	The input is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	The input group (only 1 possible)
usiInput (USINT)	The input: 1 to 2 (PMCprimo Drive) 1 to 4 (PMCprimo 2+2) 1 to 4 (PMCprimo 16+)
usiPolarity (USINT)	0: falling edge - 1: rising edge

Output variables:

bDone (BOOL)	Input has been defined (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

DisplaySnapshotPosition and SetReferenceMode

Factory setting: Input not defined

Examples:

Example in IL:

Declaration:

```
INST: DefinePositionSnapshot;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

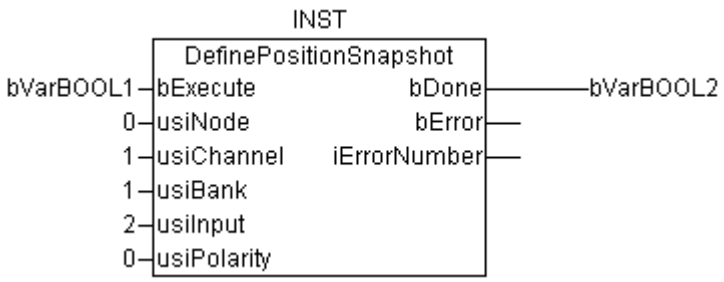
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, usiBank := 1
      ,usiInput := 2, usiPolarity := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, diPosition := 3000, usiBank := 1
      ,usiInput := 2, usiPolarity := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



DefineReferenceInput

PMCprimo-Command: DR (Define reference input)

Function library: **Referencing**

Description:

With this function block a digital input can be defined as a reference input. The axis can also be virtual. As a result this input is determined for the cycle reference correction.

Input variables:

bExecute (BOOL)	The input is defined in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiBank (USINT)	The input group (only 1 possible)
usiInput (USINT)	The input: 1 to 2 (PMCprimo Drive) 1 to 4 (PMCprimo 2+2) 1 to 4 adjustable with **CD** command (PMCprimo 16+) 1 to 4 adjustable with **CD** command (PMCprimo Drive2)
usiPolarity (USINT)	0: falling edge - 1: rising edge

Output variables:

bDone (BOOL)	Input has been defined (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetReferenceMode, UndefineInput

Factory setting: Input not defined

Examples:

Example in IL:

Declaration:

```
INST: DefineReferenceInput;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

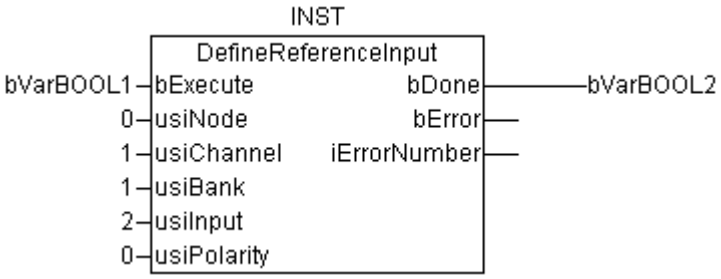
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, usiBank := 1
            , usiInput := 2, usiPolarity := 0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiBank := 1
            , usiInput := 2, usiPolarity := 0);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetFilterOnReference

PMCprimo-Command: FR (Set **f**ilter on **r**eference error)

Function library: **Referencing**

Description:

With this function block a filter can be set for a reference error. With SetFilterOnReference=0 there is no filter active. PMCprimo ignores reference errors which are bigger than SetFilterOnReference completely in case of values SetFilterOnReference ≠0. The function block is used for the filtering reference signals, which are outside of the permissible range to be expected. This module SetFilterOnReference is independent on the function block SetMaxReferenceCorrection (page 255). SetMaxReferenceCorrection limits the correction value which is corrected in case of a reference error at the motor.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiValue (UINT)	The filter value from 0 up to 65.535 increments

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetReferenceMode and SetReferenceOptionsWord

Factory setting: 0 (switched-off)

Examples:

Example in IL:

Declaration:

```
INST: SetFilterOnReference;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

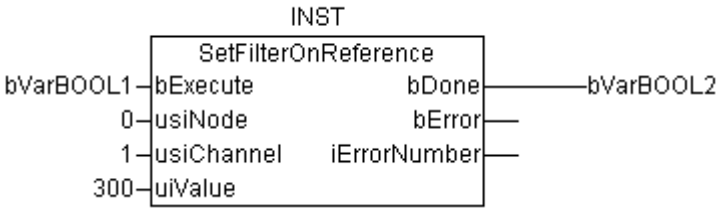
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiValue := 300)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue := 300);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetMaxReferenceCorrection

PMCprimo-Command: SR (**S**et maximum **r**eference correction)

Function library: **Referencing**

Description:

This function block limits with values unequal to zero of the reference error which is corrected at the motor to the set value. This function block can be used, to eliminate wrong reference signals which are located far away from the position to be expected or allows a cyclic referring although the reference signals do not coincide with the set position bound.

If PMCprimo recognises a reference signal the reference error is calculated as difference between the zero point defined with the zero point and the zero position or the position bound.

In case the reference error is within the set value and Bit 0 of SetReferenceOptionsWord (page 272) is 1, the reference error is corrected at the motor.

In case the reference error is outside of the set limit value, the action depends on PMCprimo on Bit 1 of the function SetReferenceOptionsWord and the value of the function SetFilterOnReference (page 253):

> In case Bit 1 is set from SetReferenceOptionsWord 0 and SetFilterOnReference to 0, the reference error is not corrected and the error message “The reference error limit has been exceeded!” is not issued. The reference error is completely ignored.
>
> In case Bit 1 of SetReferenceOptionsWord 1 and SetFilterOnReference is unequal to 0, the value of SetMaxReferenceCorrection is aligned as error maximum at the motor.

Is the reference error bigger than SetMaxReferenceCorrection, the value of the function block SetRefereneErrorLimit (page 262) zero, and Bit 1 of SetReferenceOptionsWord 1 or SetFilterOnReference unequal zero, the error signal “The reference error limit has been exceeded” is issued. It the value of SetReferenceErrorLimit is unequal to 0, this error signal is issued, when the reference error is bigger than the value of SetReferenceErrorLimit. With Bit 1 of the function block SetErrorOptionsWord (page 188) it can be determined, if the motor shall be switched-off additionally to this error signal.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiValue (UINT)	The value is written from 0 to 65.535 increments

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetReferenceOptionsWord, SetReferenceErrorLimit and SetFilterOnReference

Factory setting: 0 (switched-off)

Examples:

Example in IL:

Declaration:

```
INST: SetMaxReferenceCorrection;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

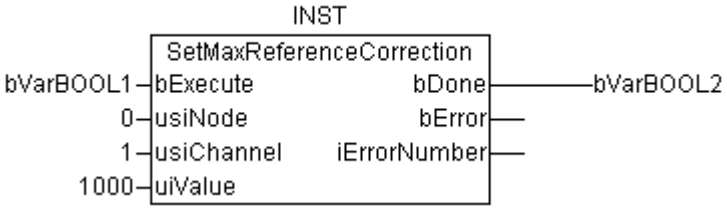
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiValue := 1000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue := 1000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetReferenceAcceleration

PMCprimo-Command: RC (Set **r**eference error adjustment a**cc**eleration)

Function library: **Referencing**

Description:

The adjustment of a reference error results in a new setpoint position for the slave axis. PMCprimo can determine the acceleration ramp for obtaining the adjustment velocity by the aid of the function block. The parameter is indicated in increments / second^2. The parameter is to be set at the slave axis. The value 0 means a jump of the compensation velocity.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
udiValue (UDINT)	The value from 0 to 2.000.000 increments/s²

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetReferenceCorrectionVelocity

Examples:

Example in IL:

Declaration:

```
INST: SetReferenceAcceleration;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

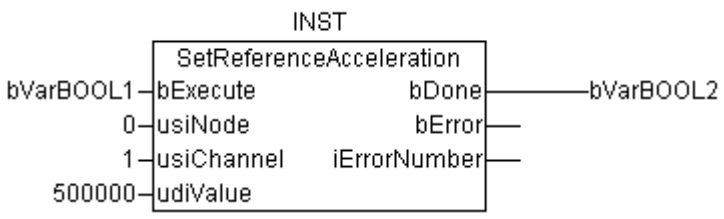
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, udiValue := 500000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, udiValue := 500000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetReferenceAdvanceFactor

PMCprimo-Command: RN

Function library: **Referencing**

Description:

This function block is used, to realise a shift dependent on the velocity of the referencing signal. This can be necessary if the used sensor switches only slowly and, therefore, the measured position depends on the velocity.

The calculation occurs according to the following formula (calculation as with PA and BA):

$$\text{Displacement} = \frac{\text{demand velocity} * RN}{65536}$$

The value is a time which results from the calculation. 1 ms corresponds to the value 64. As a result the shift can be adjusted in steps of 15,625 micro seconds.

The average of the setpoint velocity is used for the calculation. The time interval set with the function block SetVelocityAveragingTime is used.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiValue (UINT)	The value from 0 to 65535 As of version **2.005** new range: ±65.535

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetVelocityAveragingTime

Factory setting: 0 (switched-off)

Examples:

Example in IL:

Declaration:

```
INST: SetReferenceAdvanceFactor;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

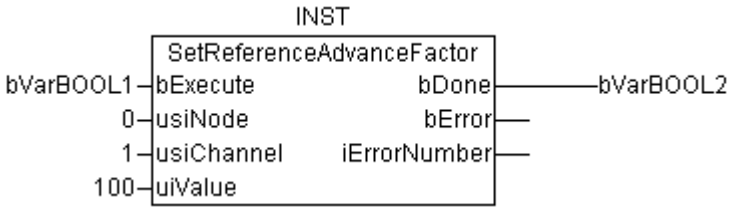
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiValue := 100)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue := 100);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetReferenceCorrectionVelocity

PMCprimo-Command: RV (Set **r**eference correction **v**elocity)

Function library: **Referencing**

Description:

The correction of a reference error requires a temporary velocity modification during the execution of a movement. PMCprimo can limit the maximum velocity during the correction of a reference error. The correction of a reference error is executed as step function when indicating zero. The velocity is indicated in % of the actual demand velocity. If a reference error is detected with on a stopped axis, the error is compensated with the velocity of SetSlowSpeed (page 236). Following connection is valid for values unequal 0:

Velocity modification = instantaneous velocity x set value/100

If PMCprimo recognises a reference signal before the existing reference error has been aligned PMCprimo signals “reference correction overrun”. This error can occur if SetMapAdjustmentVelocity is too small and very short position bounds. If Bit 2 of the function block SetErrorOptionsWord (page 188) is set, PMCprimo switches off the motor in case of an error.

The acceleration for executing a reference error (Illustration picture below) can be set with the function block SetReferenceAcceleration.

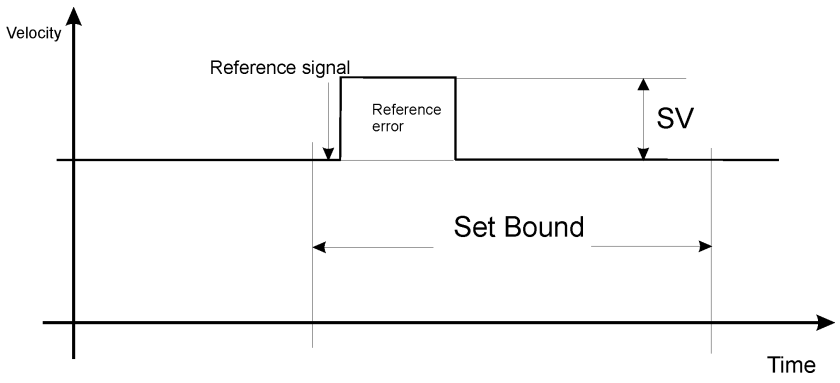


Illustration 35: Reference error correction with factor

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiValue (USINT)	The value from 0 to 200 percent

As of **Primo_V2_006.lib**: udiValue (UDINT)

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetReferenceAcceleration and SetSlowSpeed

Factory setting: 100 percent

Examples:

Declaration:

```
INST: SetReferenceCorrectionVelocity;
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

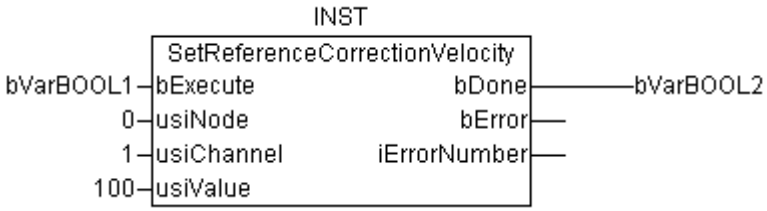
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiValue := 100)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue := 100);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetReferenceErrorLimit

PMCprimo-Command: LR (Set **r**eference error **l**imit)

Function library: **Referencing**

Description:

With this function block a reference error limit can be set in increments. If PMCprimo recognises a reference error which is bigger than the set value, the error signal "Reference error outside limits" is issued. SetReferenceErrorLimit is independent on SetMaxReferenceCorrection (page 255) and SetFilterOnReference. It can be determined with Bit 1 of the function block SetErrorOptionsWord (page 188), if the motor shall be switched off additionally to this error signal.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
udiValue (UINT)	The reference error limit from 0 (switched-off) to 65535 Increments

Ab **Primo_V2_006.lib**: uiValue (UINT)

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetErrorOptionsWord

Factory setting: 0 (switched-off)

Examples:

Declaration:

INST: SetReferenceErrorLimit
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;

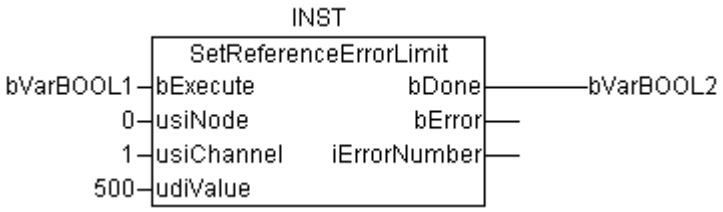
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, udiValue := 500)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, udiValue := 500);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetReferenceFalseHighLimit

PMCprimo-Command: FH (Set re**f**erence input true **h**igh limit)

Function library: **Referencing**

Description:

This function block acts in connection with the function blocks SetReferenceFalseLowLimit, SetReferenceTrueHighLimit and SetReferenceTrueLowLimit.

These four modules limit the travel length of a reference input defined with the module DefineReferenceInput. They allow that PMCprimo only reacts on a reference input, when the signal is recognised within the specified limits. The reference input is evaluated only then as valid, if it results between the values of SetReferenceFalseLowLimit and SetReferenceFalseHighLimit in untrue and afterwards between the values of SetReferenceTrueLowLimit and SetReferenceTrueHighLimit in true. The input is only evaluated as valid reference input by PMCprimo, when it has changed from true to untrue. However, the reference error for the initialisation and the cyclic reporting is determined with the rising edge (change from untrue to true) of the input signal. The values are given in increments.

An inspection of the limit values for reference input true does not occur with indication of zero SetReferenceTrueHighLimit and SetReferenceTrueLowLimit. An inspection of the limit values for a reference input untrue does not occur with indication of zero for SetReferenceFalseHighLimit and SetReferenceFalseLowLimit. With an indication of uiValue=0 at all four parameters an inspection of the limit values does not occur.

Summary:

A reference input is evaluated as such, if

- SetReferenceTrueLowLimit <=distance with reference input (DefineReferenceInput) true <= SetReferenceTrueHighLimit and
- SetReferenceFalseLowLimit <= length of travel with reference input (DefineReferenceInput) untrue <= SetReferenceFalseHighLimit

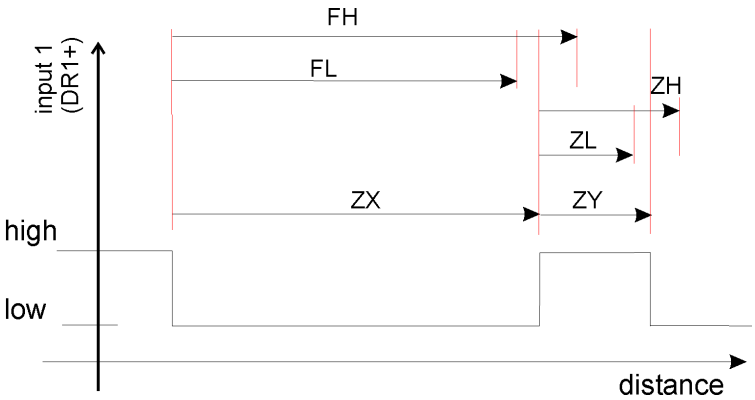


Illustration 36: Limit values for the evaluation reference input

This possibility is used, to be able to filter a valid reference signal from a spectrum of signals of the reference sensor. This filter characteristic is especially suitable for printing and registration applications.

These functions are enabled with the function blocks SetReferenceFilterOptionWord (Bit 0).

Input variables:

bExecute (BOOL) The value is written in case of a change from 0 to 1

	0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiValue (UINT)	The value from 0 to 65535 increments As of version **2.004** new range: ±2.000.000.000

Ab **Primo_V2_006.lib**: udiValue (UDINT)

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetReferenceFilterOptionWord, SetReferenceFalseLowLimit, SetReferenceFalseHighLimit, SetReferenceTrueLowLimit und SetReferenceTrueHighLimit

Factory setting: 0 (switched-off)

Examples:

Declaration:

```
INST: SetReferenceFalseHighLimit
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

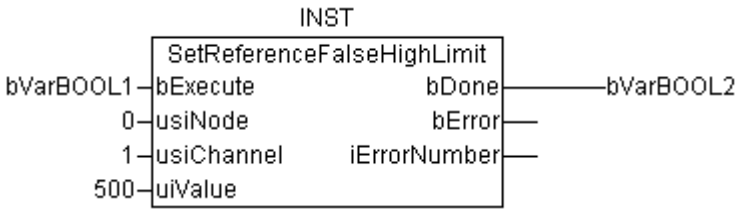
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, uiValue := 500)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, uiValue := 500);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetReferenceTrueHighLimit

PMCprimo-Command: ZH (Set reference input true **h**igh limit)

see SetReferenceFalseHighLimit on page 264

SetReferenceTrueLowLimit

PMCprimo-Command: ZL (Set reference input true low **l**imit)

see SetReferenceFalseHighLimit on page 264

SetReferenceFalseLowLimit

PMCprimo-Command: FL (Set re**f**erence input true low **l**imit)

see SetReferenceFalseHighLimit on page 264

SetReferenceFilterOptionWord

PMCprimo-Command: FW (Set reference **f**ilter options **w**ord)

Function library: **Referencing**

Description:

With this function block the options are pre-set for a reference filter of PMCprimo (Bit 0 stands for input or output on the right and Bit 7 on the left side.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiControlword (USINT)	The setting with following meaning:

Bit 0: This Bit activates the parameter SetReferenceFalseLowLimit, SetReferenceFalseHighLimit, SetReferenceTrueLowLimit und SetReferenceTrueHighLimit.

- 0: Parameter not active
- 1: Parameter active.

Bit 1: This Bit determines the reference position of the function SetReferencePosition (page 274) in PMCprimo.

- 0: PMCprimo uses the relative position to the reference signal
- 1: PMCprimo uses the absolute position of the position counter.

Bit 2: not occupied

up to

Bit 7: not occupied.

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetReferenceFalseLowLimit, SetReferenceFalseHighLimit, SetReferenceTrueLowLimit, SetReferenceTrueHighLimit und SetReferencePosition

Factory setting: 0

Examples:

Declaration:

```
INST: SetReferenceFilterOptionWord
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

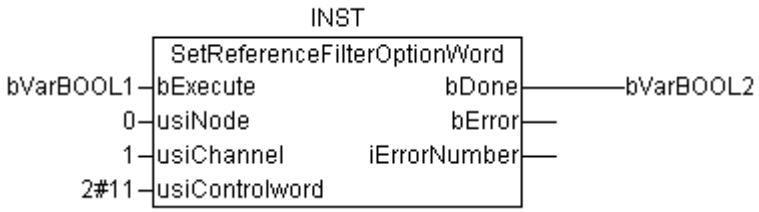
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, usiControlword := 2#11)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiControlword := 2#11);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetReferenceHoldoffTime

PMCprimo-Command: RH (Set **r**eference **h**oldoff time)

Function library: **Referencing**

Description:

With this function block a debouncing time can be set for a reference signal. PMCprimo only accepts the next reference signal after timeout of this switch-off delay.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiValue (USINT)	The filter time from 0 up to 127 milliseconds

Ab **Primo_V2_006.lib**: uiValue (UINT)

Output variables:

bDone (BOOL)	Value has been written (True) or it is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

DefineReferenceInput

Factory setting: 0 (switched-off)

Examples:

Declaration:

```
INST: SetReferenceHoldoffTime
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

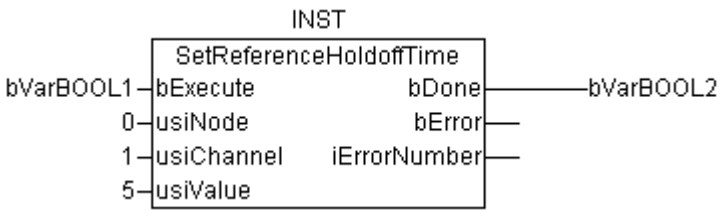
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, usiValue := 5)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiValue := 5);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetReferenceMode

PMCprimo-Command: RM (Set continuous **r**eference **m**ode on/off)

Function library: **Referencing**

Description:

Permissible values for n: 0 and 1
Factory setting: 0

This function block effects that with SetReferenceMode=1, the zero track as reference input (with function block DefineZeroMarkerInput page 69 defined) all defined reference inputs (with DefineReferenceInput page 251 defined) and all position snapshots of the encoder position of the inputs defined with the DefinePositionSnapshot (page 249) are registered in every cycle.

Input variables:

bExecute (BOOL)	The mode is set in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiValue (USINT)	The switch for referencing (0 or 1)

Output variables:

bDone (BOOL)	Mode activated (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

DefineZeroMarkerInput, DefineReferenceInput and DefinePositionSnapshot

Factory setting: 0 (switched-off)

Examples:

Declaration:

```
INST: SetReferenceMode
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

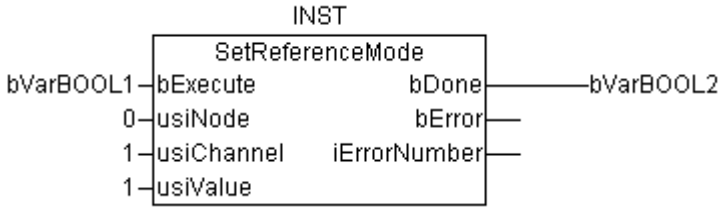
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, usiValue := 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiValue := 1);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetReferenceOffset

PMCprimo-Command: RF Set **ref**erence offset)

Function library: **Referencing**

Description:

With this function block a reference offset can be set. This is the position, at which PMCprimo expects a reference signal. The difference between the actual measured reference position and offset obtains the reference error to be corrected (see DisplayReferenceError page 58).

The set value is also used for the function blocks InitialisePosition and InitialisePositionBounds. In this case the position counter is set to the value of the reference offset when recognising the reference signal.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diValue (DINT)	The reference offset in the range of ± 4.000.000 increments

Output variables:

bDone (BOOL)	Value is written (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

DisplayReferenceError, InitialisePosition and InitialisePositionBounds

Factory setting: 0 increments

Examples:

Declaration:

```
INST: SetReferenceOffset
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

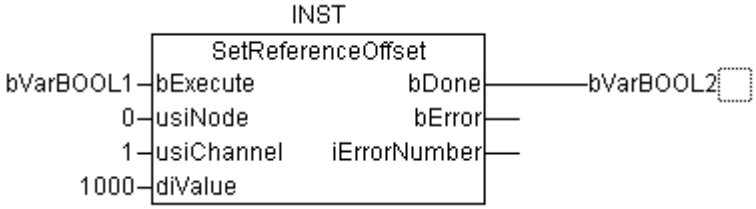
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, diValue := 1000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, diValue := 1000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetReferenceOptionsWord

PMCprimo-Command: RW (Set **r**eference options **w**ord)

Function library: **Referencing**

Description:

With this function block a function selection is done and the reference behaviour of PMCprimo is pre-set (Bit 0 stands with input or output on the right and Bit 7 on left side)

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiControlword (USINT)	The setting with following meaning:

Bit 0: This Bit activates the cyclic homing, if it is activated with the RM-command.

0: Reference correction not active.

1: Reference correction active.

Bit 1: This Bit determines the reaction of PMCprimo, if the reference error to be corrected is bigger than the permissible maximum value with reference error correction (Function SetMaxReferenceCorrection page 255).

0: PMCprimo completely ignores a reference error. Only reference signals within the window are evaluated and corrected.

1: PMCprimo executes a reference error correction with the permissible maximum value and provides the error signal “RL”.

Bit 2: This Bit activates a permissible position for correction of a reference error (setting with the function SetReferencePosition page 274).

0: A position is not defined with correction of a reference error. The correction is immediately executed when recognising a reference input.

1: Position with reference error correction active.

Bit 3: This Bit prevents the movement back to the reference signal with initialisation (Function InitialisePosition page 214).

0: Movement back to the reference signal with initialisation is executed.

1: Movement back to the reference signal with initialisation is not executed.

Bit 4: **Version 1.008 or higher:**
This bit defines the parameter of the function SetReferenceCorrectionVelocity (see page 260)

0: RV sets the velocity for the reference correction

1: RV sets the distance of the reference correction. With this function the reference error is spread out to a defined distance. Bit 6 of RW must be set if bit 4 is set. If not then a warning is given out and bit 6 is set automatic.

Bit 5: This Bit determines, if a reference error correction is only executed in the position indication or also at the motor. This Bit does not have any effect at a slave axis with active position allocation, as each reference error is corrected at the slave axis in this case.

0: Motor position and position indication are corrected.

1: Only the position indication is corrected.

Bit 6 **Version 1.008 or higher:**
This bit defines the parameter of the function SetReferenceAcceleration (see page 257)

0: RC sets the acceleration for the reference correction

1: RC sets an acceleration distance.

Bit 7: not used

Output variables:

bDone (BOOL)	Value is written (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetReferenceMode, SetMaxReferenceCorrection, SetReferencePosition and InitialisePosition

Factory setting: 0

Examples:

Declaration:

```
INST: SetReferenceOptionsWord
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

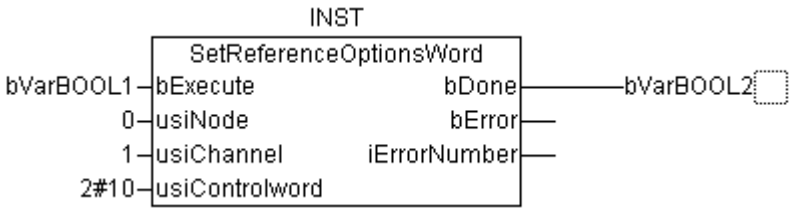
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, usiControlword := 2#10)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiControlword := 2#10);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetReferencePosition

PMCprimo-Command: RJ (Set deferred **r**eference ad**j**ustment position)

Function library: **Referencing**

Description:

With this function block the user can define the position within the cycle limits, at which PMCprimo may correct a stored reference error. In case the value is zero, then PMCprimo immediately executes the correction of the reference error. The position, which can be specified with this command, must be activated with Bit 2 of the pre-adjustment of the reference behaviour SetReferenceOptionsWord (page 272). Only a positive position may be indicated.

In case a value is specified n-times bigger than SetPositionBound (page 133) or SetReferenceRepeatLength (page 278), if used, the reference error correction is executed, displaced by n-cycles. Bit 1 of SetReferenceFilterOptionWord (page 267) must be set to 1, as a relative position indication does not make any sense in this case.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diValue (DINT)	The reference position in the range of ± 4.000.000 increments

Output variables:

bDone (BOOL)	Value is written (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetReferenceOptionsWord, SetPositionBound, SetReferenceRepeatLength and SetReferenceFilterOptionWord

Factory setting: 0 Increments

Examples:

Declaration:

INST: SetReferencePosition
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;

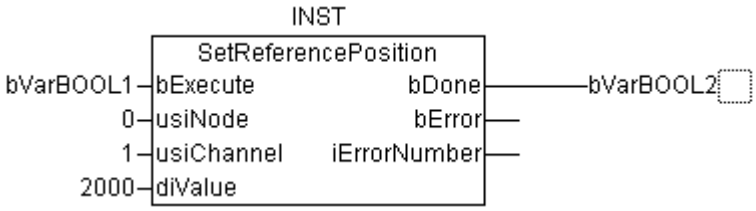
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, diValue := 2000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, diValue := 2000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetReferencePositionAtOtherPosition

PMCprimo-Command: RK

Function library: **Referencing**

Description:

This function block effects that the reference error correction of the actual axis is executed at the position of another axis. The actual axis waits until the other axis has reached the specified position and starts only then with the reference error correction. The function block SetReferencePosition (page 274) is not effective any more on the actual axis.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiAtNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiAtChannel (USINT)	The axis number from 1 to n (depending on the system)
diValue (DINT)	The position of the other axis in the range of ± 4.000.000 increments

Output variables:

bDone (BOOL)	Value is written (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetReferenceOptionsWord, SetReferenceMode

Factory setting: not defined

Examples:

Declaration:

INST: SetReferencePositionAtOtherPosition
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;

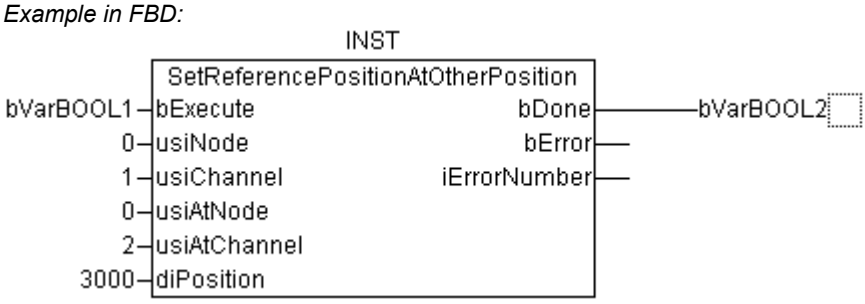
Example in IL:

```
CAL    INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiAtNode := 0,
usiAtChannel := 2, diPosition := 3000)
LD     INST.bDone
ST     bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiAtNode := 0, usiAtChannel := 2,
      diPosition := 3000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetReferenceRepeatLength

PMCprimo-Command: RL (Set **r**eference repeat **l**ength)

Function library: **Referencing**

Description:

With this function block a reference position bound can be set. The reference position bound indicates the position, at which PMCprimo expects a reference signal. If zero is set, then PMCprimo expects a reference signal at the cycle limit (set with the function block SetPositionBound page 133).

Application example:

The position bound of the machine amounts to 10 000 increments and the reference position bound is set with 2 500 increments. As a result PMCprimo corrects 4 times a reference error within a position bound of the machine.

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
udiValue (UDINT)	The value of 0 to 4.000.000 increments, 0 switched-off As of version **2.004** new range: 0 to 2.000.000.000

Output variables:

bDone (BOOL)	Value is written (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetPositionBound

Factory setting: 0 (switched-off)

Examples:

Declaration:

```
INST: SetReferenceRepeatLength
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

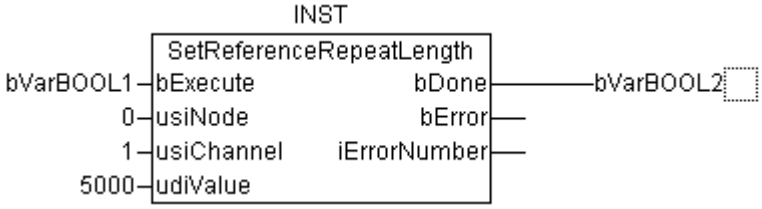
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, udiValue := 5000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, udiValue := 5000);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



SetReferenceTimeout

PMCprimo-Command: RT (Set **r**eference **t**imeout)

Function library: **Referencing**

Description:

With this function block a monitoring can be activated for reference signals. If a value unequal to zero is set, reference signals can get lost, before PMCprimo issues the error signal „reference timeout“ If Bit 0 of the module SetErrorOptionsWord (page 188) is set to 1, PMCprimo simultaneously switches off the controller enable (motor is de-energised). Herewith the initialisation (InitialisePosition, InitialisePositionBounds page 214, 216) can also be monitored. The monitoring is only used for signals which are not filtered out (function block SetFilterOnReference page 253).

Input variables:

bExecute (BOOL)	The value is written in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiValue (USINT)	The number of signals, which can get lost from 0 (switched-off) to 255

Output variables:

bDone (BOOL)	Value is written (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetErrorOptionsWord and SetFilterOnReference

Factory setting: 0 (switched-off)

Examples:

Declaration:

INST: SetReferenceTimeout
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;

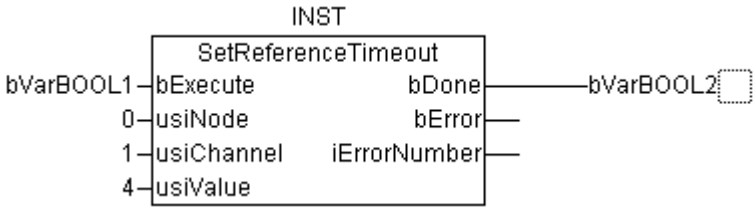
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel := 1, usiValue := 4)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel := 1, usiValue := 4);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



UndefineInput

PMCprimo-Command: UI (**U**ndefine **i**nput definition)

Function library: **Referencing**

Description:

With this function block the user resets a defined input function in PMCprimo, independent of the fact which input function was defined up to now.

Input variables:

bExecute (BOOL)	The input function is reset in case of a change from 0 to 1 0 resets the function block and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiBank (USINT)	The input group from 1 to 3
usiInput (USINT)	The input from 1 to 8

Output variables:

bDone (BOOL)	Definition reset (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

DefinePositionSnapshot and DefineReferenceInput

Examples:

Declaration:

```
INST: UndefineInput
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

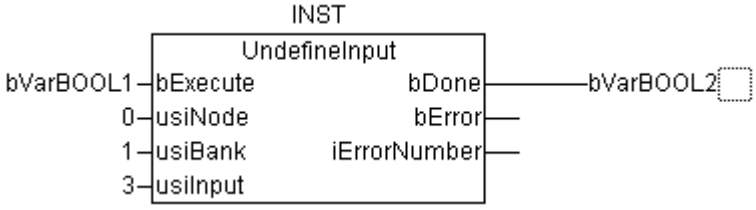
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiBank:= 1, usiInput:= 3)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiBank:= 1, usiInput:= 3);
bVarBOOL2:=Inst.bDone;
```

Example in FBD:



14.16 Tension Control

With the web tension control (tension), winding-up processes can be realised for example.

The value of the analogue input is decisive for the control loop of the web tension. The value of the analogue input is read at regular intervals (real time) and from that a transmission ratio between the master and slave axes is calculated. The algorithm for this calculation is:

$$i = SM + SM \times [AP\ e_i + AI\ \Sigma e_i + AD\ (e_i - e_{i-1})]$$

i	=	Transmission ratio Master/Slave
SM	=	Set transmission ratio (SM-command)
AP	=	Proportional factor
AI	=	Integral factor
AD	=	Differential factor
e_i	=	Web tension distance (= set web tension- actual web tension)

The dynamic behaviour of the web tension depends on these constant factors and on the mechanical behaviour of the driven machine. The setting of these factors is imperative for the achievement of an optimum control behaviour.

CalculateInitialRatio

PMCprimo-Command: CR (**C**alculate initial **r**atio from analogue range distances)

Function library: **Tension**

Description:

As soon as the function module is started, PMCprimo calculates the initial transmission ratio from the values of the slave and master axes specifed with SetAnalogueRangeDistance and SetAnalogueMasterRangeDistance. The calculation is executed with both measuring values with SetAnalogueRangeDistance (Slave)/SetAnalogueMasterRangeDistance (Master) in the same way as with the values of SetPositionBound by the function SetMapScaleFromBounds.

In case the automatic measuring of the analogue distance is released at the master and slave axes, the function module ExecuteMap executes an automatic measuring of the analogue distance at both axes.

Input variables: :

bExecute (BOOL)	The transmission ratio is calculated in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL) (False)	Transmission ratio calculated (True) or module is still under operation
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetAnalogueRangeDistance and SetAnalogueMasterRangeDistance

Examples:

Declaration:

INST: CalculateInitialRatio
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;

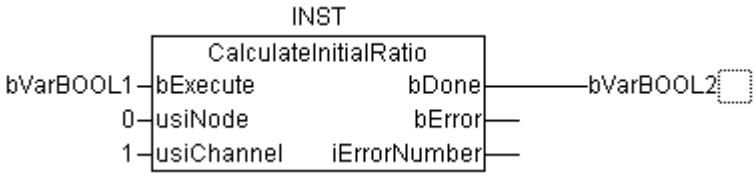
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



ExecuteAnalogueDistanceInit

PMCprimo-Command: XR (E**x**ecute analogue **r**ange distance initialisation)

Function library: **Tension**

Description:

The function "Initialisation of the transmission ratio with analogue distances" can manually be started. The measuring results are registered in the SetAnalogueRangeDistance parameter.

The command is always set down at the slave axis.

The "Initialisation of the transmission ratio with analogue distances" functions as follows:

The motor moves in the same direction with the velocity (function module SetSlowSpeed), as with the start-up of the web tension demand value, the position stores and stopps until an analogue limit value has been exceeded. Subsequently the motor moves in the opposite direction, these second position stores until an analogue value has been exceeded again and stores the distance between these both values as value of SetAnalogueRangeDistance for the slave.

The Bits 4 and 5 in the function SetAnalogueControlWord release the "Initialisation of the transmission ratio with analogue distances", i.e. this function is automatically executed when the position allocation is activated with ExecuteMap. The determined transmission ratio SetAnalogueRangeDistance (Slave)/SetAnalogueMasterRangeDistance (Master) is used as initial transmission ratio (see function module CalculateInitialRatio).

Input variables:

bExecute (BOOL)	The transmission ratio is determined in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiMasterSlave (USINT)	0: distance is measured on slave, 1: distance is measured on master

Output variables:

bDone (BOOL)	Transmission ratio determined (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetAnalogueRangeDistance, SetAnalogueMasterRange Distance and CalculateInitialRatio

Examples:

Declaration:

INST: ExecuteAnalogueDistanceInit
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;

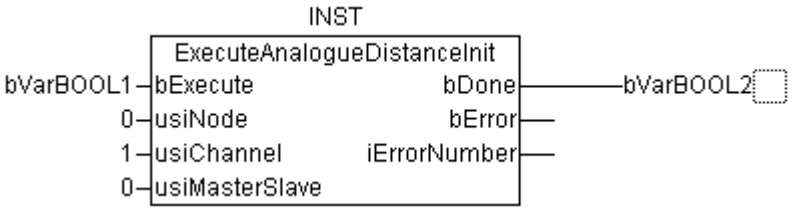
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1, usiMasterSlave:=0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1, usiMasterSlave:=0);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



SetAnalogueControlMode

PMCprimo-Command: AM (Set **a**nalogue control **m**ode)

Function library: **Tension**

Description:

This function module releases an analogue control. The value 1 means to release web tension control , 0 blocks the analogue control. The analogue control can be released or blocked at any time.

The activation of a position allocation with the function module ExecuteMap initiates various actions at the slave axis. In case of an enabled web tension control these actions are defined by pre-setting of the web tension with the function module SetAnalogueControlWord. The Bits 0, 1 and 4 of the module SetMapOptionsWord are compulsory set to 1 by the release of the web tension control. It is a must for the function of the software clutch and the velocity ratio in the web tension control circuit.

In case Bit 4 or Bit 5 of SetAnalogueControlWord are set to 1, PMCprimo automatically initialises the transmission ratio, before the position allocation is activated for the web tension control. PMCprimo measures the distance between both positions, where the limit values for the analogue input are exceeded or fall below, by moving the master/slave axes. The transmission with these measured values obtains a good estimation for the transmission ratio with an activation of the web tension control. This values allows PMCprimo, to achieve the transmission ratio to be driven considerably faster than using the pre-set transmission ratio (SetScaleMapping), especially then, when the machine does not start from the initial position.

Winding-up or unwinding is an example for this application. Normally the machine starts with a full or empty coil and the initial transmission ratio is specified by the function module SetScaleMapping. A restart after an interruption takes place smoothly, as a transmission ratio determined at last is used for the start. If the machine, however, is started with partially winded or unwinded coils with unknown diameters, the transmission ratio normally specified is not suitable for the new coil diameter. The machine can be started by this automatic initialisation of the transmission ratio at the beginning of the web tension control without a big settling process of the web tension control circuit of the pre-set ratio to the actually required transmission ratio.

Input variables:

bExecute (BOOL)	The mode of operation is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiMode (USINT)	The mode of operation: 0 switched off, 1 switched on

Output variables:

bDone (BOOL)	Mode of operation set (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetAnalogueControlWord and ExecuteMp

Factory setting: 0 (switched off)

Examples:

Declaration:

```
INST: SetAnalogueControlMode
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

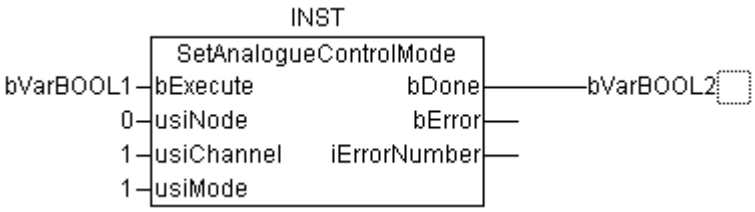
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1, usiMode:=1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1, usiMode:=1);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



SetAnalogueControlSetPoint

PMCprimo-Command: AC (Set **a**nalogue **c**ontrol setpoint)

Function library: **Tension**

Description:

With this function module the setpoint value of the analogue control is specified. The control deviation is calculated from the difference between the setpoint value and the actual value measured with the analogue input.

Input variables:

bExecute (BOOL)	The setpoint is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
iSetPoint (INT)	The web tension setpoint value in the range of ±2047 (corresponds to ±10V)

Output variables:

bDone (BOOL)	Setpoint set (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetAnalogueControlWord, SetAnalogueControlMode and Execute Map

Factory setting: 0

Examples:

Declaration:

```
INST: SetAnalogueControlSetPoint
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

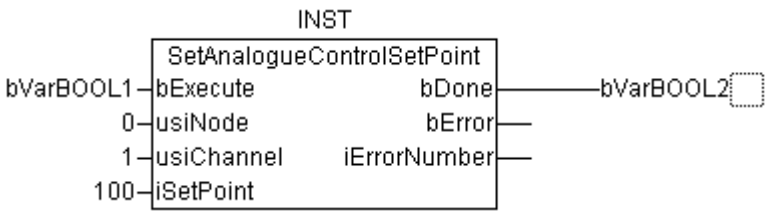
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1, iSetPoint:=100)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1, iSetPoint:=100);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



SetAnalogueControlWord

PMCprimo-Command: AW (Set **a**nalogue control options **w**ord)

Function library: **Tension**

Description:

With this function module the behaviour of the analogue control is defined. (Bit 0 is the right and bit 7 the left one)

Bit 0: This Bit determines the behaviour of the slave axis with activation of a position allocation with Execute Map.

0: The slave axis immediately goes into the condition of the active position allocation under use of the software clutch SetClutchTime to the transmission ration specified by the web tension.

1: The slave axis drives with the jog speed (set with SetSlowSpeed) until the web tension achieves the web tension setpoint value and afterwards goes into the condition of the active position allocation.

Bit 1: Reserved.

Bit 2: This Bit determines the mode of operation of the integral factor. The web tension depends on the movement of the master and slave axes, as the output of the control loop of the velocity ratio is between both axes. If the master axis stands, then also the slave axis stands and a control of the velocity ratio is possible. In this case it is necessary, to block the I-part, to prevent that a big web tension distance adds itself and effects a step response during start of the motor.

0: The integral factor is blocked.

1: The integral factor is released.

Bit 3: This Bit determines, with which transmission ratio the web tension control is activated.

0: The initial transmission ratio is the value of the module SetScaleMapping.

1: The initial transmission ratio is the actual transmission ratio. Normally it is, the instantaneous transmission ratio determined the last before stop of the slave axis.

Bit 4: This Bit releases the automatic initialisation between the upper and lower limit value at the slave axis during activation of a position allocation with a web tension control (see also SetAnalogueRangeDistance and ExecuteAnalogueDistanceInit for detail information).

0: The automatic initialisation of the transmission ratio with the upper and lower limit values is not released at the slave axis.

1: The automatic initialisation of the transmission ratio with the upper and lower limit values is released at the slave axis.

Bit 5: This Bit releases the automatic initialisation between the upper and lower limit values at the master axis during activation of a position allocation with web tension control (see also SetAnalogueRangeDistance and ExecuteAnalogueDistanceInit for Detail information). This function is also possible, when the slave and master axes are in a linked system on different controls.

0: The automatic initialisation of the transmission ratio with the upper and lower limit value is not released at the master axis.

1: The automatic initialisation of the transmission ratio with the upper and lower limit value is released at the master axis.

Bit 6: This Bit determines the starting direction of the web tension setpoint value, when

Bit 0 is set to 1 and the web tension control is released.

0: The starting direction is the same as the sign of the web tension distance.

1: The starting direction is opposite the sign of the web tension distance.

Bit 7: This Bit determines the sign of the analogue control.

0: An increase of the error signal (web tension distance) effects a voltage rise the setpoint output.

1: An increase of the error signal (web tension distance) effects a voltage reduction at the setpoint output.

Input variables:

bExecute (BOOL)	The value is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
usiControlword (USINT)	The control word for setting the behaviour:

Output variables:

bDone (BOOL)	Setpoint set (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetAnalogueControlMode and ExecuteMap, SetScaleMapping, SetClutchTime and SetSlowSpeed

Factory setting: 0

Examples

Declaration:

```
INST: SetAnalogueControlWord
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

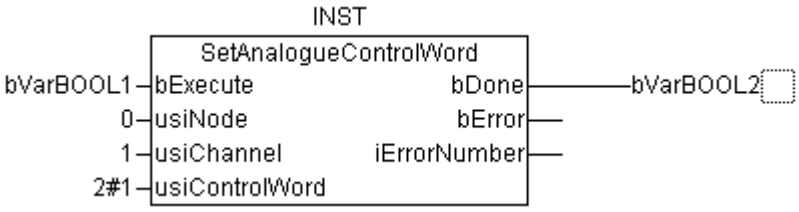
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1, usiControlWord:=2#1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1, usiControlWord:=2#1);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



SetAnalogueDifferentialGain

PMCprimo-Command: AD (Set **a**nalogue **d**ifferential control gain)

Function library: **Tension**

Description:

With this function module a differential factor of the control algorithm of the analogue control is set. This factor uses the differential of the actual value distance which shows the control deviation of the system. This factor is useful with very fast modifications of the control deviation and acts in a damping way on the analogue control.

Input variables:

bExecute (BOOL)	The value is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiGain (UINT)	The control factor from 0 up to 65535

Output variables:

bDone (BOOL)	Value is set (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions

SetAnalogueIntegralGain, SetAnalogueProportianalGain und SetAnalogueControlMode

Factory setting: 0 (switched off)

Examples:

```
INST: SetAnalogueDifferentialGain
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

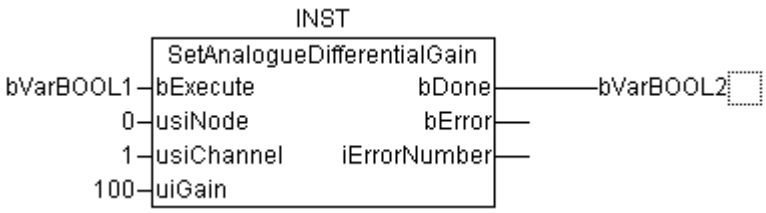
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1, uiGain:=100)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1, uiGain:=100);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



SetAnalogueInputHighLimit

PMCprimo-Command: AH (Set **a**nalogue input **h**igh limit)

Function library: **Tension**

Description:

With this funciton module a permissible upper limit can be set for the analogue input. In case the value of the analogue input exceeds this value, PMCprimo issues a corresponding error signal. If the Bit 3 of the module SetErrorOptionsWord is set to 1, the exceeding of the upper limit is evaluated as motor-OFF-error.

Input variables:

bExecute (BOOL)	The value is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diLimit (DINT)	The limit value of –2.147.483.648 up to 2.147.483.647 (2047 corresponds to 10V, it the analogue input is used)

Output variables:

bDone (BOOL)	Value is set (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions

SetAnalogueInputLowLimit und SetAnalogueControlMode

Factory setting: 2000

Examples:

Example in IL:

```
INST: SetAnalogueInputHighLimit
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

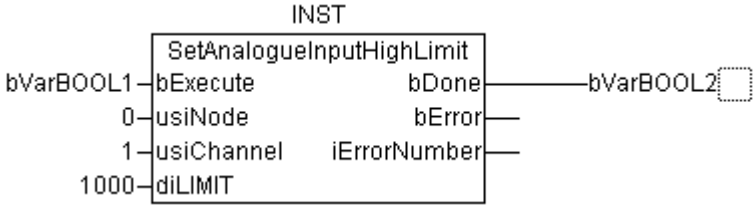
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1, diLimit:=1000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1, diLimit:=1000);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



SetAnalogueInputLowLimit

PMCprimo-Command: AL (Set **a**nalogue input **l**ow limit)

Function library: **Tension**

Description:

With this funciton module a permissible lower limit can be set for the analogue input. In case the value of the analogue input falls below of this value, PMCprimo issues a corresponding error signal. If the Bit 3 of the module SetErrorOptionsWord is set to 1, the exceeding of the lower limit is evaluated as motor-OFF-error.

Input variables:

bExecute (BOOL)	The value is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diLimit (DINT)	The limit value of –2.147.483.648 up to 2.147.483.647 (2047 corresponds to 10V, it the analogue input is used)

Output variables:

bDone (BOOL)	Value is set (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions

SetAnalogueInputHighLimit und SetAnalogueControlMode

Factory setting: -2000

Examples:

```
INST: SetAnalogueInputLowLimit
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

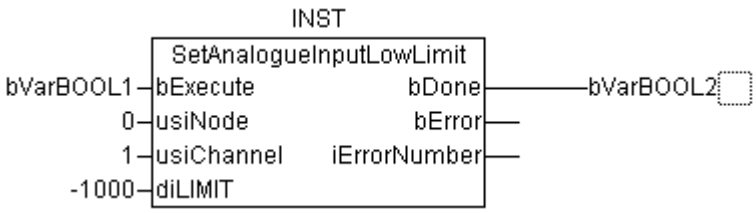
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1, diLimit:=-1000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1, diLimit:=-1000);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



SetAnalogueIntegralGain

PMCprimo-Command: AI (Set **a**nalogue **i**ntegral control gain)

Function library: **Tension**

Description:

With this function module the integral factor of the control algorithm of the analogue control is set. When using an integral factor PMCprimo integrates the control deviation by adding the actual error to a consecutive overall error. The integral factor is mainly used for winding and unwinding applications with web tension control.

If the integral component is used you have to observe that during standstill of the axis this one is switched off with the function module SetAnalogueControlWord, as otherwise it is started with an incorrect transmission ratio when restarting and the material to be unwinded is damaged.

Input variables:

bExecute (BOOL)	The value is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiGain (UINT)	The control factor from 0 to 65535

Output variables:

bDone (BOOL)	Value is set (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions

SetAnalogueDifferentialGain, SetAnalogueProportianalGain und SetAnalogueControlMode

Factory setting: 0 (switched off)

Examples:

INST: SetAnalogueIntegralGain
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;

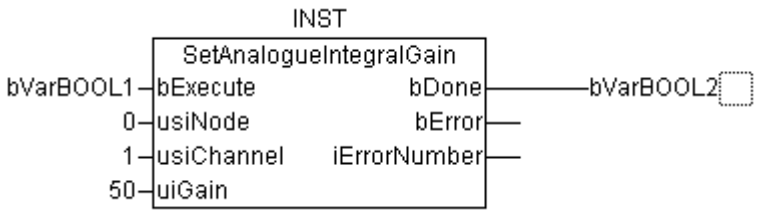
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1, uiGain:=50)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1, uiGain:=50);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



SetAnalogueMasterRangeDistance

PMCprimo-Command: MM (Define **m**aster axis analogue range distance)

Function library: **Tension**

Description:

This function module determines the distance between the upper and lower limits of the master axis, called master analogue distance. It allows, to specify the value of the analogue distance for a master axis.

Input variables:

bExecute (BOOL)	The value is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiDistance (UINT)	The distance from 0 to 65535

Output variables:

bDone (BOOL)	Value is set (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetAnalogueRangeDistance und SetAnalogueControlMode

Factory setting: 256 increments

Examples:

```
INST: SetAnalogueMasterRangeDistance
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

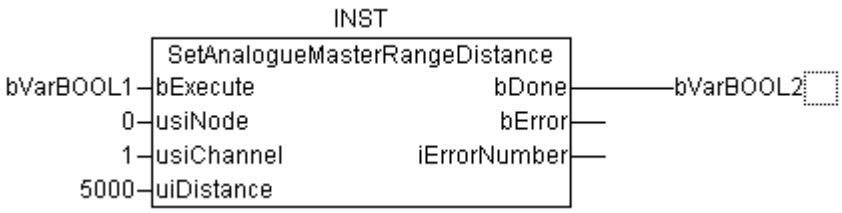
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1, uiDistance:=5000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1, uiDistance:=5000);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



SetAnalogueProportianalGain

PMCprimo-Command: AP (Set **a**nalogue control **p**roportional gain)

Function library: **Tension**

Description:

With this function module the proportional factor of the control algorithm is set for the analogue control. A big value of the AP-factor allows a short reaction time and an exact analogue control. Therefore, the AP-factor should be set as high as possible without generating an overshoot.

Input variables:

bExecute (BOOL)	The value is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiGain (UINT)	The control factor from 0 to 65535

Output variables:

bDone (BOOL)	Value is set (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions

SetAnalogueDifferentialGain, SetAnalogueIntegralGain und SetAnalogueControlMode

Factory setting: 10.000

Examples:

```
INST: SetAnalogueProportianalGain
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

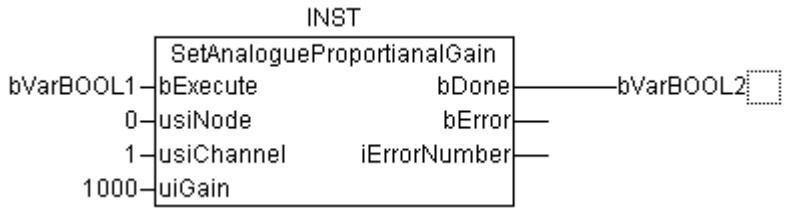
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1, uiGain:=1000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1, uiGain:=1000);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



SetAnalogueRangeDistance

PMCprimo-Command: AR (Define **a**nalogue **r**ange distance)

Function library: **Tension**

Description:

This function module determines the distance between the upper and lower limits of the analogue input of the instantaneous selected axis, called analogue distance. It allows, to specify the value of the analogue distance for a slave axis.

Input variables:

bExecute (BOOL)	The value is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
uiDistance (UINT)	The distance at the slave from 0 to 65535 increments

Output variables:

bDone (BOOL)	Value is set (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Connected functions:

SetAnalogueMasterRangeDistance und SetAnalogueControlMode

Factory setting: 256 increments

Examples:

```
INST: SetAnalogueMasterRangeDistance
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

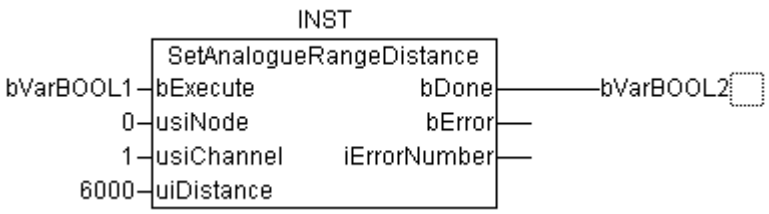
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1, uiDistance:=6000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1, uiDistance:=6000);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



14.17 Wait

With the wait command it is possible to wait for a specific motor status or position.

WaitEndState

PMCprimo-Command: WE (**W**ait **e**nd **s**tate)

Function library: **Wait**

As of **version 2.004** available

Description:

This command ends the current wait state as completed normally. This allows the user to escape from a wait state

Input variables:

bExecute (BOOL)	The value is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	No more waiting (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Examples:

```
INST: WaitEndState
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

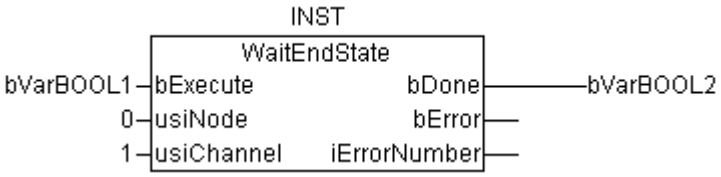
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



WaitForAbsolutePosition

PMCprimo-Command: WA (**W**ait for **a**bsolute position)

Function library: **Wait**

Available as of version 2.000

Description:

This function block tells PMCprimo to wait until the current channel reaches the given absolute position . The position is specified in increments.

Input variables:

bExecute (BOOL)	The value is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diValue (UINT)	Absolute position for waiting.

Output variables:

bDone (BOOL)	No more waiting (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Examples:

INST: WaitForAbsolutePosition
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;

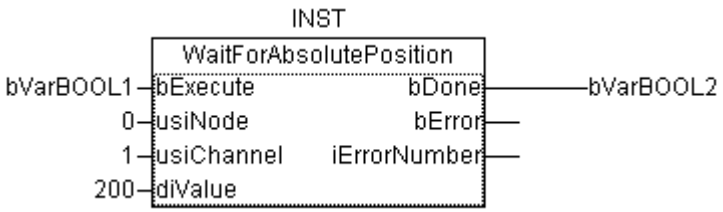
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1, diValue:=200)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1, diValue:=200);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



WaitForBoundPosition

PMCprimo-Command: WB (**W**ait for **b**ound position)

Function library: **Wait**

Available as of version 2.000

Description:

This function block tells PMCprimo to wait until the motor passes the next bound position (positive or negative direction)

Input variables:

bExecute (BOOL)	The value is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)

Output variables:

bDone (BOOL)	No more waiting (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Examples:

INST: WaitForBoundPosition
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;

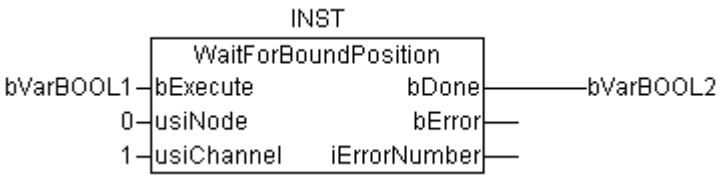
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



WaitForRelativePosition

PMCprimo-Command: WR (**W**ait for **r**elative position)

Function library: **Wait**

Available as of version 2.000

Description:

This function block tells PMCprimo to wait until it reaches the specified position relative to some previous position. The position is specified in increments.

The function block WaitForRelativePosition starts a position counter. If the position counter reaches the value the function block bDone is set to TRUE.

Input variables:

bExecute (BOOL)	The value is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diValue (UINT)	Relative position for waiting.

Output variables:

bDone (BOOL)	No more waiting (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Example:

```
INST: WaitForRelativePosition
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;
```

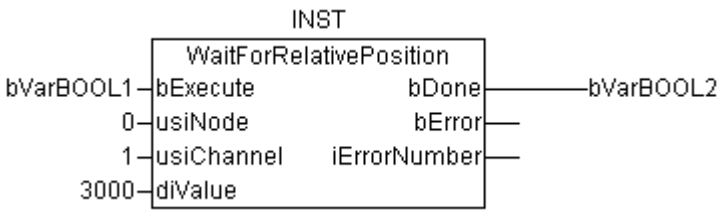
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1, diValue:=3000)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1, diValue:=3000);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



WaitForStatusMotor

PMCprimo-Befehl: WS (**W**ait for motor **s**tatus)

Function library: **Wait**

Available as of version 2.000

Description:

INST.bDone changes to TRUE if the specified channels goes to state given by INST.diValue. bVarBool1 must be equal to TRUE, during the command is executed.

Input variables:

bExecute (BOOL)	The value is set in case of a change from 0 to 1 0 resets the function module and the output variables are set to False or 0
usiNode (USINT)	The node number in the linked system, 0 if only one unit or Host
usiChannel (USINT)	The axis number from 1 to n (depending on the system)
diValue (UINT)	Motor status for waiting.

Output variables:

bDone (BOOL)	No more waiting (True) or module is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Example:

INST: WaitForStatusMotor
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;

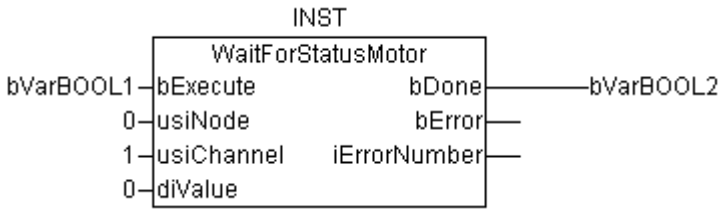
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1,usiNode := 0, usiChannel:= 1, diValue:=0)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiNode := 0, usiChannel:= 1, diValue:=0);
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



SendMail

PMCprimo-command is not available

Function library: **Tension**

As of **version 2.000** available

Description:

With this function block an email can be send.

The system has to be connected with Ethernet to the internet and the IP address of the mail server has to be known. This is possible with the command ping from windows (example ping exchange -> result 10.10.1.20). The mail server is different in every company. It's possible to use free mail servers in the internet if a firewall is not used to prevent the access (port 25).

After bExecute is set TRUE the email will be send.

Input variables:

bExecute (BOOL)	Avter change from 0 to 1 the email will be send. 0 resets the function module and the output variables are set to False or 0
sServer (STRING)	mail server address
sFromAddress (STRING)	sender of email
sToAddress (STRING)	reciever of email
sSubject (STRING)	subject text
sMessageText (STRING)	message text

Output variables:

bDone (BOOL)	Mail was send successfully (True) or mail sending still in progress (False)
bError (BOOL)	True: an error occured; False: no error
iErrorNumber (INT)	Gives the excact error cause (see GetError page 203)

Example:

INST: SendMail
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;

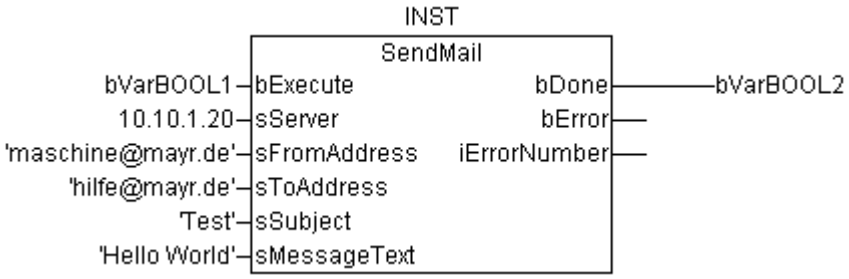
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, sServer:='10.10.1.20',
      sFromAddress:='maschine@pilz.de', sToAddress:='hilfe@pilz.de'
      sSubject:='Test', sMessageText:='Hello World')
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, sServer:='10.10.1.20',
      sFromAddress:='maschine@pilz.de', sToAddress:='hilfe@pilz.de'
      sSubject:='Test', sMessageText:='Hello World');
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



ExecuteSequence

PMCprimo-Command: XS sequence (E**x**ecute **s**equence)

Function library: **Tension**

Description:

This function module enables the user to call a PMCprimo sequence. A sequence call effects the immediate execution of the indicated sequence. In case of an error the handling of the sequence is immediately stopped. A sequence may only be recalled after the complete execution (in case of non-observance PMCprimo gives the error message „Cannot execute sequence while it is in use.“. Commands are executed one after another.

Input variables:

bExecute (BOOL)	The sequence is started in case of a change from 0 to 1 0 resets the function module and the output variables > are set to False or 0
sName (STRING)	Name of th sequence

Output variables:

bDone (BOOL)	sequence executed (True) or sequence is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Examples:

INST: ExecuteSequence
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;

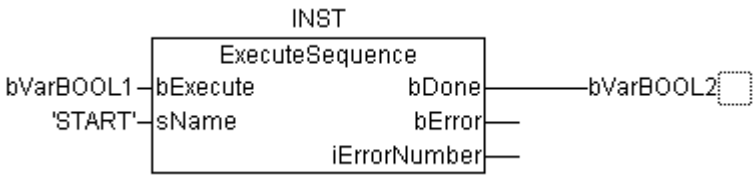
Example in IL:

```
CAL   INST(bExecute := bVarBOOL1, sName:='START')
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, sName:='START');
bVarBOOL2:=Inst.bDone;
```

Example in FDB:



SetModbusAddress

PMCprimo-Befehl: MU

As of **version 2.006** available

Description:

This function module sets the modbus number. This is used in a ModBus multidrop net, to indentify the device.

Input variables:

bExecute (BOOL)	If the value is set the function starts working. 0 resets the function module and the output variables are set to False or 0
usiAddress (USINT)	The modbus number in the range of values 0 to 255

Output variables:

bDone (BOOL)	sequence executed (True) or sequence is still under operation (False)
bError (BOOL)	True: an error has occurred; False: no error
iErrorNumber (INT)	Indicates the exact error cause (see GetError page 203)

Example:

INST: SetModbusAddress
bVarBOOL1: BOOL;
bVarBOOL2: BOOL;

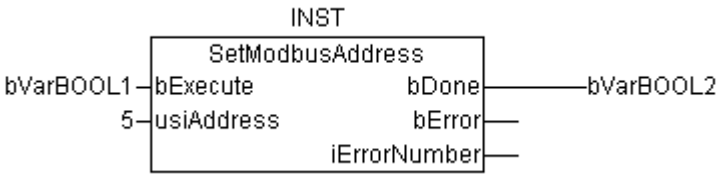
Example in AWL:

```
CAL   INST(bExecute := bVarBOOL1, usiAddress:=5)
LD    INST.bDone
ST    bVarBOOL2
```

Example in ST:

```
INST(bExecute := bVarBOOL1, usiAddress:=5);
bVarBOOL2:=Inst.bDone;
```

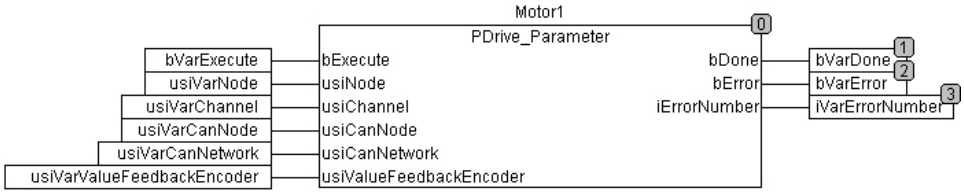
Example in FUP:

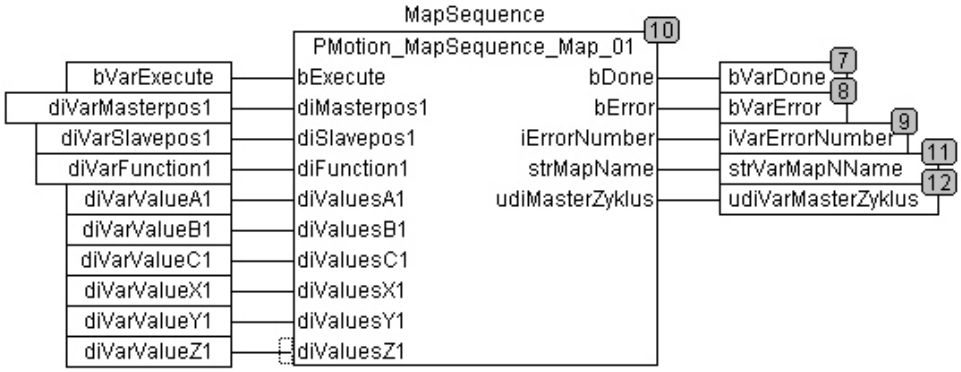


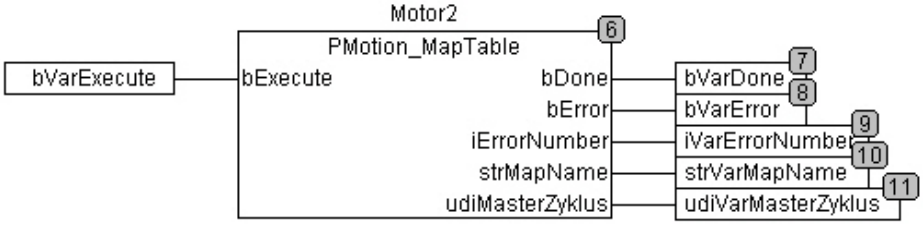
15 Function library primo_tools.lib

The functional library primo_tools.lib contains 3 function blocks, which can be included in the function editor (CFC editor). These function blocks combine the functionality of PDrive and PMotion with CoDeSys.

Examples:







The function blocks of the primo_tools.lib library are described in a separate manual: "CoDeSys and Motion Control Tools data exchange via primo.dll".

16 Cross reference list PMCprimo-Commands – SoftPLC Functions

PMCprimo-Command	SoftPLC-Function	Page
$Bx.x (read)	GetBusVariable	41
$Bx.x (write)	SetBusVariable	43
$Fx.x	GetError	203
$Fx.x=0	ResetError	225
$Sx.x	GetStatus	212
PMCtendo DD4-Command	DriveCommand	66
AA	SetAlignmentAcceleration	109
AB	AbortMotor	201
AC	SetAnalogueControlSetPoint	291
AD	SetAnalogueDifferentialGain	295
AE	DefineAnalogueLimitErrorOutput	141
AH	SetAnalogueInputHighLimit	296
AI	SetAnalogueIntegralGain	298
AL	SetAnalogueInputLowLimit	297
AM	SetAnalogueControlMode	289
AO	DefineAuxiliaryOutput	35
AP	SetAnalogueProportianalGain	300
AR	SetAnalogueRangeDistance	301
AV	SetMapAdjustmentVelocity	117
AW	SetAnalogueControlWord	292
BA	SetMapBaseAdvance	119
BC	SetPositionOverflowCounter	135
BL	SetBacklashDistance	229
BM	SetModbusAddressSetModbusAddressSetModbusAddress	310
BO	DefineBoundOverflowOutput	143
BR	SetMapScaleFromBounds	132
BT	SetMapBaseAdvanceTimeConstant	121
CI	SetClutchWindow	115
CL	SetClutchLength	111
CR	CalculateInitialRatio	285
CT	SetClutchTime	113
CW	SetControlWord	182
DA	DisplayAnalogueInput	53
DC	SetDeceleration	232
DD	DisplayDemandPosition	54
DE	DefineMotorErrorOutput	145
DF	DisplayReferenceError	58
DP	DisplayActualPosition	52
DS	DisplaySnapshotPosition	63
DV	DisplayVelocity	64
DZ	DefineZeroMarkerInput	69
EM	DefineMap	90
EW	SetErrorOptionsWord	188
FC	SetEncoderFeedbackChannel	70
FE	DisplayFollowingError	55
FH	SetReferenceFalseHighLimit	264
FL	SetReferenceFalseLowLimit	266
FR	SetFilterOnReference	253
FS	SetFeedbackEncoder	76
FW	SetReferenceFilterOptionWord	267

GF	GlobalOff	177
GM	GetMappedMasterBound	96
GS	GlobalStop	178
GW	GetWrapAroundOffset	98
IB	InitialisePositionBounds	216
ID	InitialiseDemandOffset	179
IN	InitialisePosition	214
JB	DefineReferenceBackwardOutput	153
JH	DefineReferenceForwardOutput	155
KA	SetAccelerationFeedForwardGain	181
KD	SetDifferentialGain	186
KF	SetVelocityFeedForwardGain	196
KI	SetIntegralGain	187
KM	SetMonitorOutputGain	39
KP	SetProportionalGain	185
KV	SetVelocityFeedbackGain	195
LH	SetHighPositionLimit	190
LL	SetLowPositionLimit	191
LW	SetMapLinkOptionsWord	125
MA	MoveToAbsolutePosition	222
MB	SetMapBaseOffset	123
MF	SetSlaveMapOffset	138
ML	MapLinkSlaveToMaster	104
MM	SetAnalogueMasterRangeDistance	201
MO	MotorOff	180
MR	MoveRelativePosition	220
MR	SetAnalogueMasterRangeDistance	299
MS	SetEncoderScaling	73
MT	SetMapPositionTimeout	131
MW	SetMapOptionsWord	127
NB	SetNumberOfBits	83
NL	MapLinkSlaveToDifferentialMaster	102
OM	SetMonitorOutputOffset	40
OR	DefineReferenceOutput	157
OW	DefineOutsideWindowOutput	147
PA	SetPhaseAdvanceFactor	166
PC	EnablePositionControl	176
PD	CANPositionControlDrive und CANNetworkPositionControlDrive	45
PD	CANPositionControlDrive	45
PH	SetPositionOutputHysteresis	170
PO	DefinePositionTriggerOutput	149
PS	DefinePositionSnapshot	249
PT	SetEncoderFilterTime	72
QR	GetCanSDO	48
QS	SetCanSDO	50
RA	DefineReferenceAcceptedOutput	151
RC	SetReferenceAcceleration	257
RF	SetReferenceOffset	271
RH	SetReferenceHoldoffTime	269
RI	DefineReferenceInput	251
RJ	SetReferencePosition	274
RK	SetReferencePositionAtOtherPosition	276
RL	SetReferenceRepeatLength	278
RM	SetReferenceMode	270

RN	SetReferenceAdvanceFactor	258
RR	DefineReferenceRejectOutput	159
RT	SetReferenceTimeout	280
RV	SetReferenceCorrectionVelocity	260
RW	SetReferenceOptionsWord	272
SA	SetAcceleration	227
SB	SetPositionBound	133
SC	SetCreepDistance	230
SE	SetMaxPositionError	192
SF	SetMonitorOutputFunction	37
SM	SetScaleMapping	136
SR	SetMaxReferenceCorrection	255
SR	SetReferenceErrorLimit	262
SS	SetSlowSpeed	236
ST	StopMotor	242
STnn	StopMotorToPosition	244
SV	SetVelocity	240
SW	SetWindow	200
TI	SetTimeoutForWindow	194
TM	TransferMapData	139
TO	SetEncoderTimeout	74
UI	UndefineInput	282
UL	UnlinkSlaveToMaster	140
UO	UndefineOutput	173
VC	MoveConstantVelocity	218
VH	SetVelocityOutputHysteresis	172
VJ	SetSlowSpeedMode	238
VM	SetVirtualMotorMode	198
VO	DefineVelocityOutput	164
VT	SetVelocityAveragingTime	65
WE	WaitEndState	302
WA	WaitForAbsolutePosition	303
WB	WaitForBoundPosition	304
WR	WaitForRelativePosition	305
WS	WaitForStatusMotor	306
XA	SetAbortDecelaration	226
XM	ExecuteMap	92
XR	ExecuteAnalogueDistanceInit	287
XS	ExecuteSequence	309
XV	ExecuteMapVirtual	94
XX	LengthOfAlignmentMove	100
ZC	SetPositionCounter	193
ZH	SetReferenceTrueHighLimit	266
ZL	SetReferenceTrueLowLimit	266
ZW	SetMoveOptionsWord	234
ZX	DisplayReferenceLengthFalse	59
ZY	DisplayReferenceLengthTrue	61

17 Index

T

U

V

W

...

In many countries we are represented by our subsidiaries and sales partners.

Please refer to our Homepage for further details or contact our headquarters.

www

www.pilz.com

Technical support

+49 711 3409-444

Pilz GmbH & Co. KG
Sichere Automation
Felix-Wankel-Straße 2
73760 Ostfildern, Germany
Telephone: +49 711 3409-0
Telefax: +49 711 3409-133
E-Mail: pilz.gmbh@pilz.de

pilz

more than automation

safe automation

21 471-06, 2007-03 Printed in Germany