Albatros

3.1.9

Manufacturer's manual

Tecnologie e Prodotti per l'Automazione

Table of Contents

1 System Configuration

1.1 Introduction

In the chapter concerning the composition of the system, we have already seen how the Albatros system consists of one or more modules forming a plant and how each one of these is organised in a hierarchical structure.
To configure the machine from the point of view of Albatros it is necessary to follow a sequence of operations which enable to configure the various logic levels and the underlying hardware. The general order to be followed when configuring a system is:

- Module Configuration
- Definition of Groups and Subgroups
- Devices Configuration
- Machine Configuration
- System Configuration
- Hardware Configuration
- Virtual physical Configuration

Basically Module, Group and Machine Configuration determine the logic structure of the machine, while the System, Hardware and Physical Virtual Configuration determine the physical structure.
We will analyze each one of these points in detail in the following paragraphs.

1.2 Devices Configuration

1.2.1 Introduction

In the chapter concerning the composition of the Albatros system, we described the various types of devices which can appear in a module. Now we will describe the devices from the point of view of their configuration.

Each type of device can be configured a maximum number of times, as specified in the following list:

Type of device	**Max. number**
Analog input	128
Analog output	128
Digital input	4096
Digital output	4096
Output Nibble	256
Input Nibble	256
Input port	512
Output port	512
Axis	240
Timer	128
Counter	128
Flag Bit	1024
Flag Switch	256
Flag Port	256
Function	4096

The data to be specified during configuration depends on the device, however, except for axis devices, it is almost always the same. We will now see the configuration of certain devices.

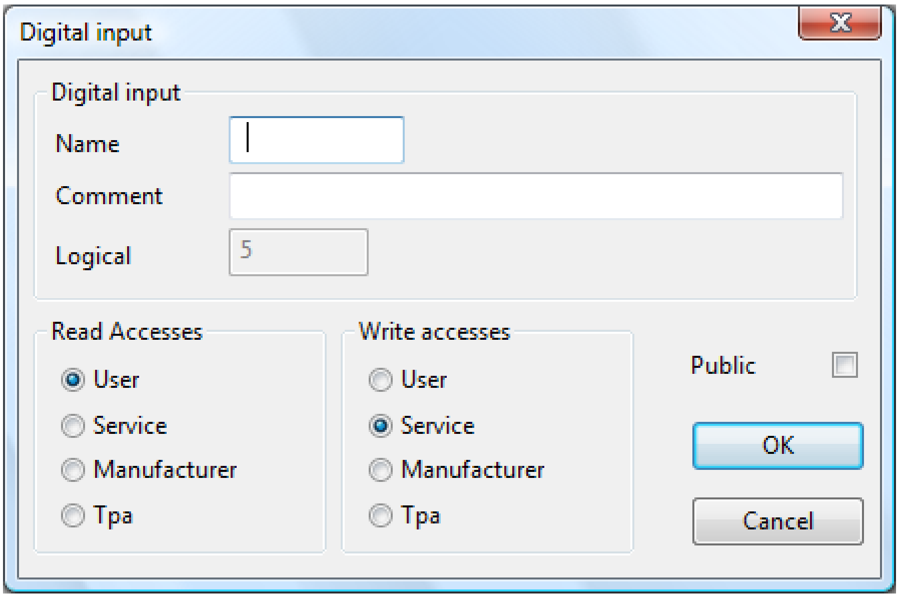
IMPORTANT:
To a card **Can** can be associated only devices of type: digital input, digital output, input port, output port.

1.2.2 Generic Device

Most devices require the same configuration parameters. Below we have illustrated the configuration of a Digital Input, however the same considerations apply to:

- Flag bit
- Flag switch
- Analog output
- Input Port

- Flag Port
- Timer
- Counters
- Input Nibble
- Output Nibble
- Function



Digital input

Digital input

Name

Comment

Logical 5

Read Accesses

● User

○ Service

○ Manufacturer

○ Tpa

Write accesses

○ User

● Service

○ Manufacturer

○ Tpa

Public □

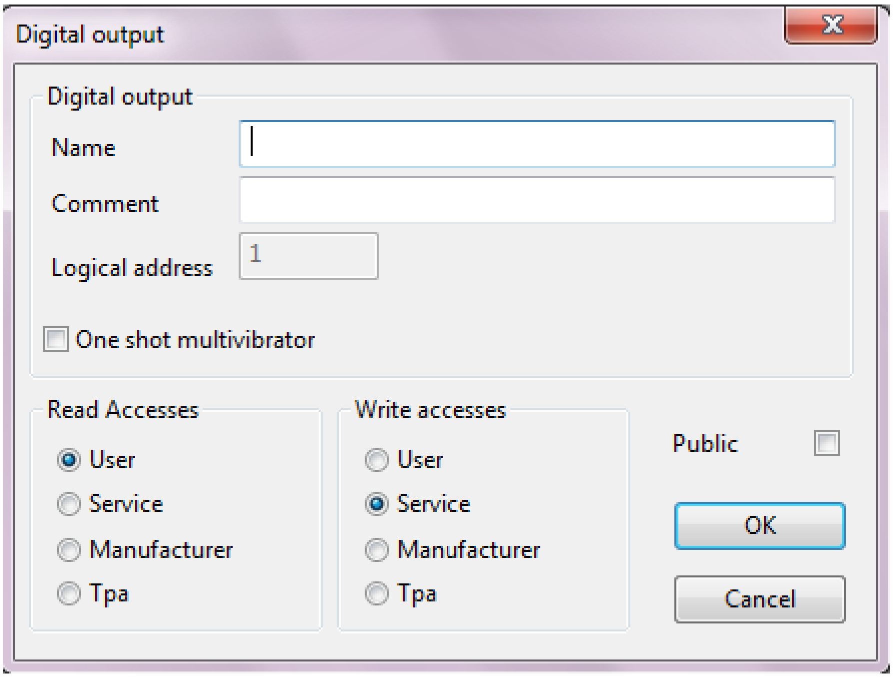
General Input configuration window

To configure any device among those listed above, the following settings must be specified:

- **Name:** name of the device, a maximum of 40 characters.
- **Comment:** a brief description of the device, it can be translated into various languages, no spaces.
- **Logical:** assigned automatically by the system.
- **Read Accesses:** specifying the minimum access level required for the device to be visualised in the Diagnostic windows or in the Synoptic Views.
- **Write accesses:** specifying the minimum access level required to modify the state of the device.
- **Public:** specifying if the state of the device can be read or modified by a GPL code not belonging to the group of the device.

1.2.3 Digital output

The digital output has one parameter that standard devices do not have: the *One shot multivibrator*



Digital output

Digital output

Name

Comment

Logical address 1

☐ One shot multivibrator

Read Accesses

● User

○ Service

○ Manufacturer

○ Tpa

Write accesses

○ User

● Service

○ Manufacturer

○ Tpa

Public ☐

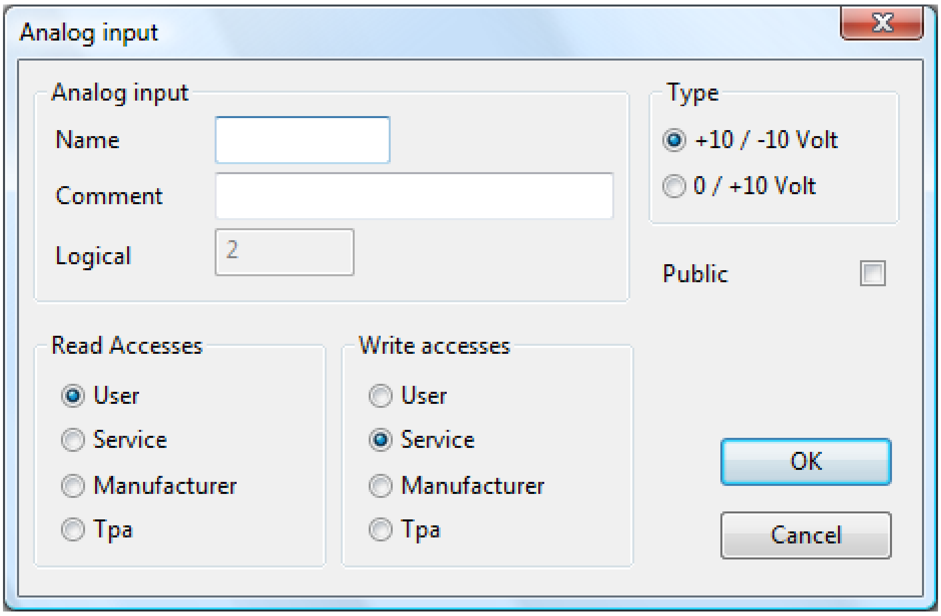
Digital output configuration window

To configure a digital output, the following settings must be specified:

- **Name:** name of the device, a maximum of 40 characters.
- **Comment:** a brief description of the device, it can be translated into various languages, no spaces.
- **Logical:** assigned automatically by the system.
- **One shot multivibrator:** if selected, it configures the output as one shot multivibrator, which means that when the output is set to ON it switches automatically back to OFF 200 ms later.
- **Read Accesses:** specifying the minimum access level required for the device to be visualised in the Diagnostic windows or in the Synoptic Views
- **Write accesses:** specifying the minimum access level required to modify the state of the device.
- **Public:** specifying whether the state of the device can be read or modified by a GPL code not belonging to the group of the device.

1.2.4 Analog input

The analog input has one parameter that standard devices do not have: *the type of power in input.*



Analog input

Analog input

Name

Comment

Logical 2

Type

● +10 / -10 Volt

○ 0 / +10 Volt

Public □

Read Accesses

● User

○ Service

○ Manufacturer

○ Tpa

Write accesses

○ User

● Service

○ Manufacturer

○ Tpa

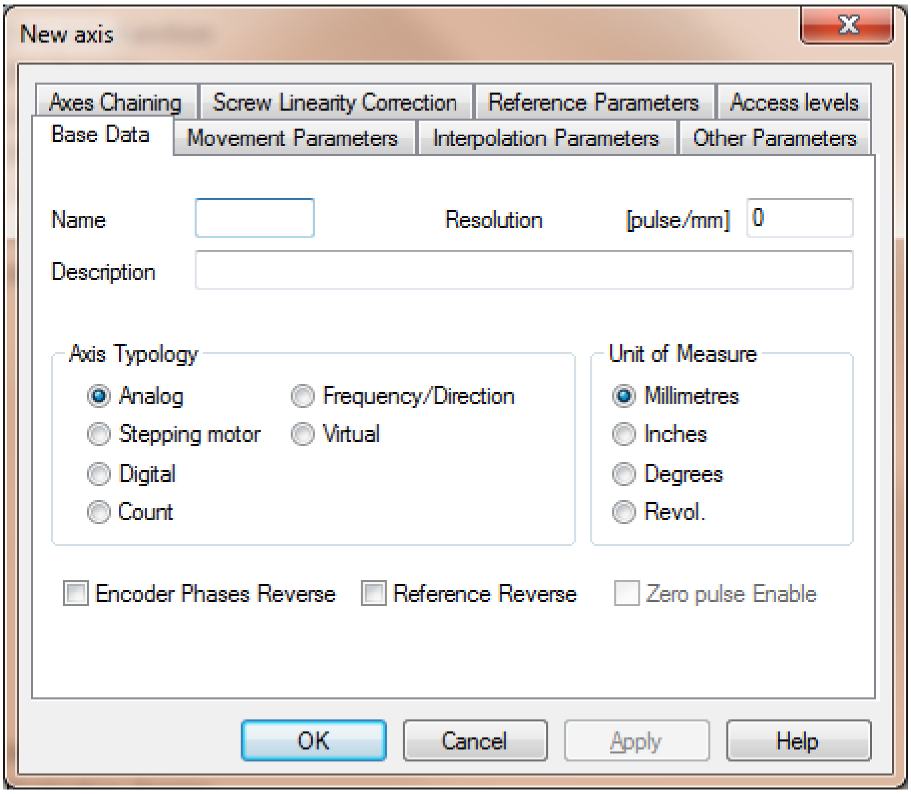
Analog input configuration window

To configure an Analog input the following settings must be specified:

- **Name:** name of the device, a maximum of 40 characters.
- **Comment:** a brief description of the device, it can be translated into various languages, no spaces.
- **Logical:** assigned automatically by the system.
- **Type**: to select the power interval read in input.
- **Read Accesses:** specifying the minimum access level required for the device to be visualised in the Diagnostic windows or in the Synoptic Views
- **Write accesses:** specifying the minimum access level required to modify the state of the device.
- **Public:** specifying if the state of the device can be read or modified by a GPL code not belonging to the group of the device.

1.2.5 Axis

Base Data



Name

Resolution [pulse/mm] 0

Description

Axis Typology

● Analog ○ Frequency/Direction

○ Stepping motor ○ Virtual

○ Digital

○ Count

Unit of Measure

● Millimetres

○ Inches

○ Degrees

○ Revol.

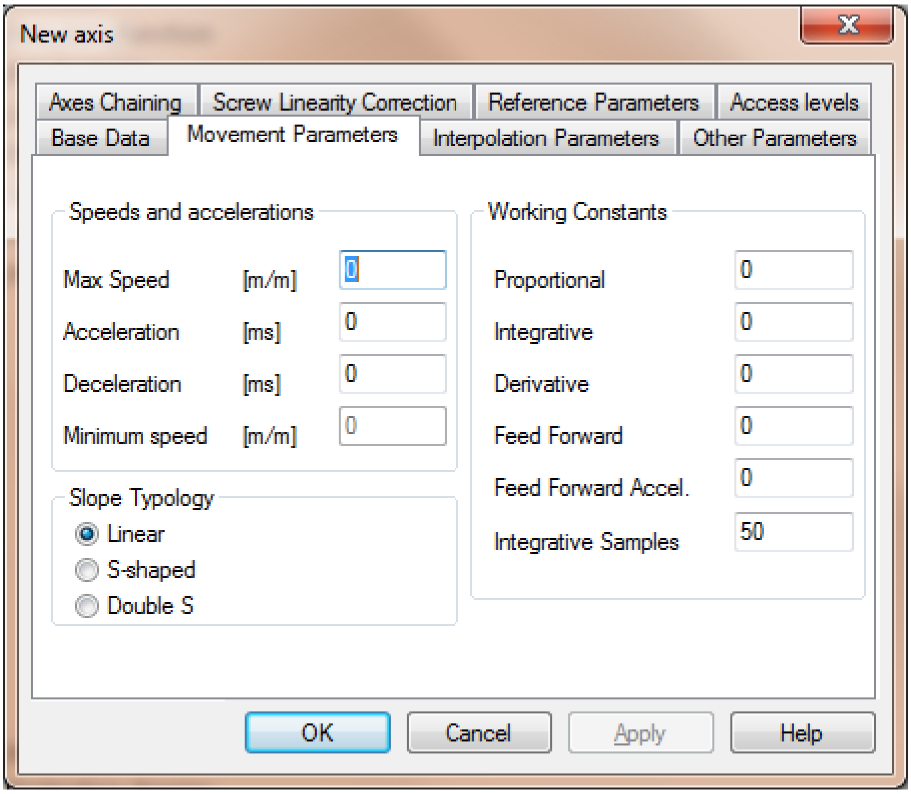
□ Encoder Phases Reverse □ Reference Reverse □ Zero pulse Enable

The base data to be specified is:

- **Name:** name of the device, a maximum of 40 characters.
- **Description:** a brief description of the device, which can be translated into various languages, no spaces.
- **Resolution:** resolution of the encoder, depending on the characteristics of the encoder and on the specified unit of measure . Remember that Albatros axis cards count as pulses the rising edges and the falling edges of both encoder phases (a 2500 pulses/revolution encoder will be detected as a10000 pulses /revolution encoder).
- **Axis Typology:** type of axis. The types are: ***Analog*** (analogically controlled), ***Stepping motor***, ***Digital***, ***Count*** (only encoder reading), ***Frequency/Direction*** (4° AlbSLM card connector), ***Virtual***.
- **Unit of Measure:** the unit of measure used to indicate the position of the axes. As all the derived dimensions depend on it, we advise to set this parameter before any other.
- **Encoder Phases Rev:** it allows you to correct via software a possible cable inversion of the encoder phases.
- **Reference Reverse:** it allows you to reverse the speed reference of the axis. If used with the encoder phases reverse it allows you to reverse the direction of the axis (if cabling is correct).
- **Zero pulse enable:** only available for counting axes, it automatically resets the position to zero when the encoder pulse is detected.

Movement parameters

Parameters used for axis point to point movement.



Speeds and accelerations

Max Speed	[m/m]	0
Acceleration	[ms]	0
Deceleration	[ms]	0
Minimum speed	[m/m]	0

Slope Typology

● Linear
○ S-shaped
○ Double S

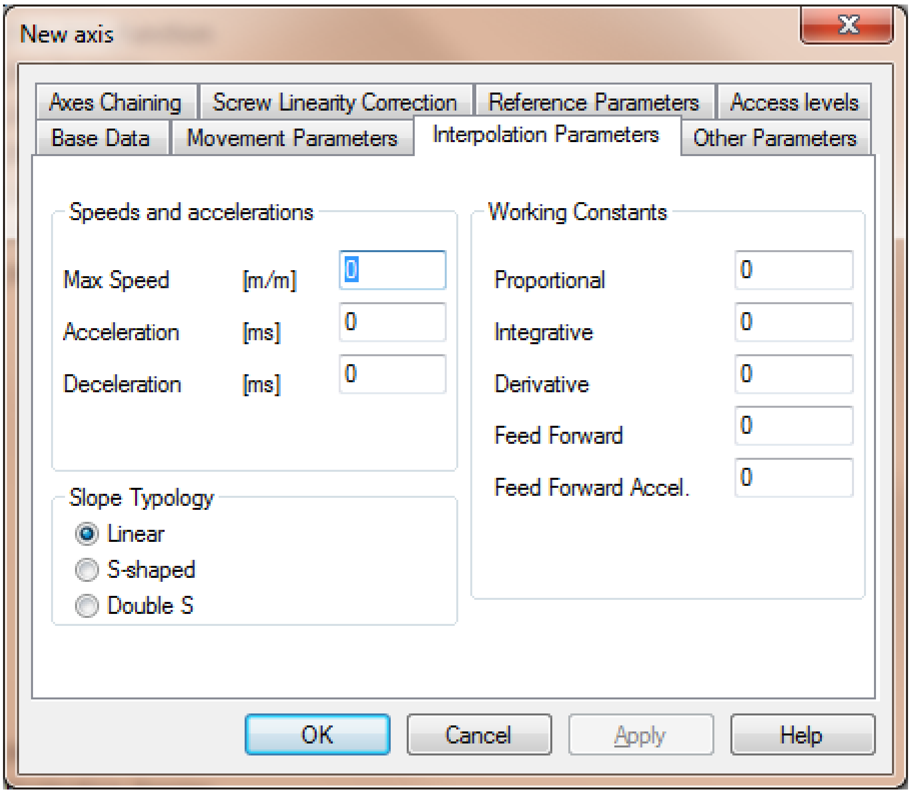
Working Constants

Proportional	0
Integrative	0
Derivative	0
Feed Forward	0
Feed Forward Accel.	0
Integrative Samples	50

- **Max Speed:** max speed of the axis.
- **Acceleration:** time of acceleration ramp.
- **Deceleration:** time of deceleration ramp.
- **Minimum Speed:** speed reached by the motor in a single step;it can only be set on stepping motor axes.
- **Slope Typology:** ramp typology of acceleration and deceleration. Not available for stepping motors.
- **Proportional:** proportional coefficient of the position loop PID controller.
- **Integrative:** integration coefficient of the position loop PID controller.
- **Derivative:** derivation coefficient of the position loop PID controller.
- **Feed Forward:** percentage of feed forward. It allows you to reduce the loop error at equal speed.
- **Feed Forward Accel.:** percentage of feed forward acceleration. It allows you to eliminate the remaining loop error (not eliminated by the feed forward) during axis acceleration and deceleration phases.
- **Integrative Samples:** Sets the number of samples of loop error, used to calculate the integral component. Valid values are in the range 1 to 200. The default value is 50. See gpl SETINTEGTIME instruction.

Interpolation parameters

Parameters used for axis interpolation movement.



Speeds and accelerations

Max Speed [m/m] 0

Acceleration [ms] 0

Deceleration [ms] 0

Slope Typology

● Linear

○ S-shaped

○ Double S

Working Constants

Proportional 0

Integrative 0

Derivative 0

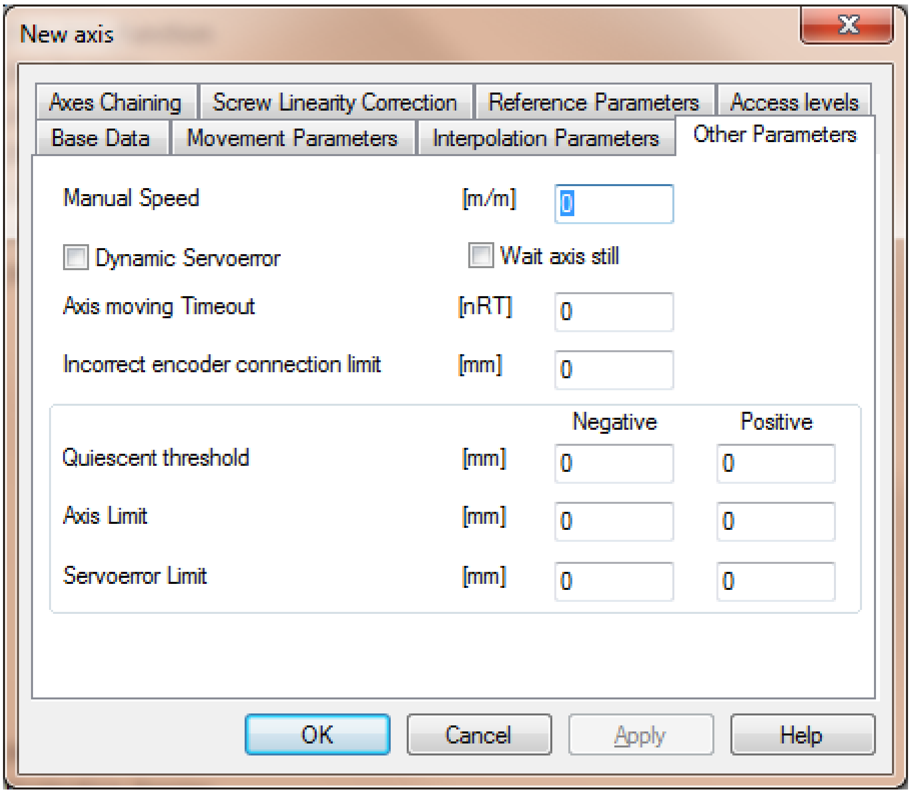
Feed Forward 0

Feed Forward Accel. 0

Except for minimum speed, they have the same meaning as the parameters described in the Moving Parameters. However these are used for interpolation movements.

Note: acceleration and deceleration values, set in the interpolation parameters, cannot be lower than the corresponding values in the movement parameters.

Other parameters



Manual Speed [m/m] 0

□ Dynamic Servoerror □ Wait axis still

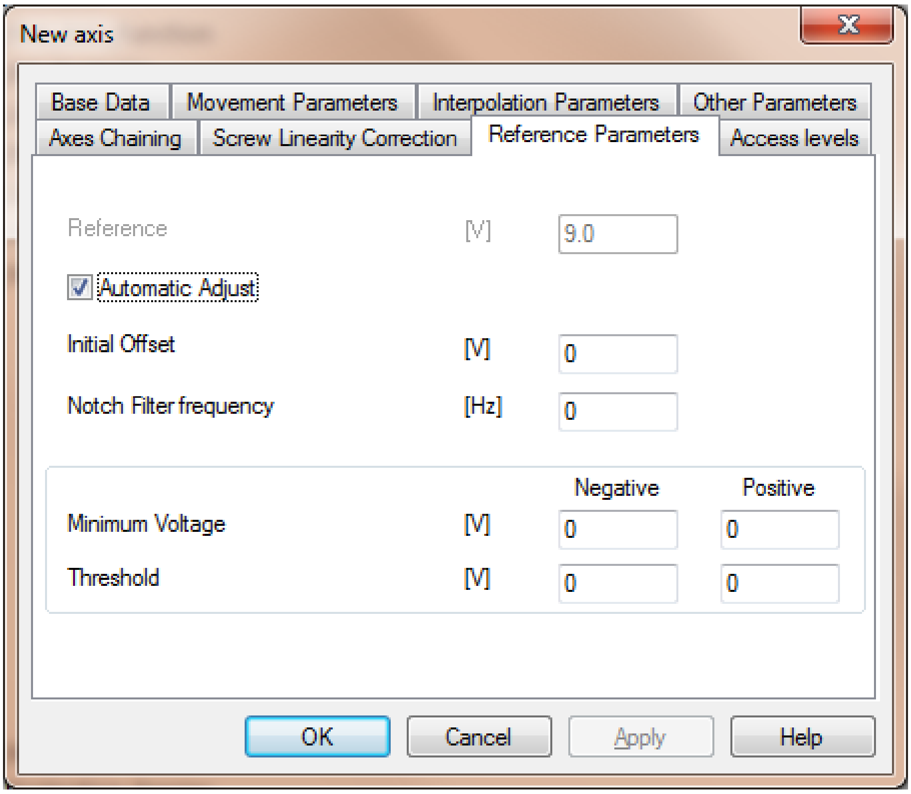
Axis moving Timeout [nRT] 0

Incorrect encoder connection limit [mm] 0

		Negative	Positive
Quiescent threshold	[mm]	0	0
Axis Limit	[mm]	0	0
Servoerror Limit	[mm]	0	0

- **Manual Speed:** specifying the maximum configuration speed which can be used in manual movements. It will never exceed the maximum set speed.
- **Dynamic Servoerror:** Valid values are 0 (= normal) and 1 (= dynamic). The default value is 0. See gpl SETMAXERTYPE instruction.
- **Wait axis still**: enables or disables the overshoot recovery function. It sets a pause of 50 ms at the end of each movement.
- **Axis moving timeout:** Valid values are in the range 0 to 1024. See gpl ENABLESTARTCONTROL instruction.
- **Incorrect encoder connection limit:** The set values are expressed in the unit of measure that axis resolution is expressed in. The settable values must be in the range 128/axis resolution to 16384/axis resolution. The default setting is calculated based on a number of steps equivalent to 1024, i.e. 1024/axis resolution.
- **Positive Limit for Servoerror:** maximum value of the loop error for loop correction in positive direction.
- **Negative Limit for Servoerror:** maximum value of the loop error for loop correction in negative direction.
- **Positive Axis Limit:** maximum value of axis running in positive direction.
- **Negative Axis Limit:** maximum value of axis running in negative direction.
- **Positive quiescent threshold:** tolerance on arrival position in positive direction.
- **Negative quiescent threshold:** tolerance on arrival position in negative direction.

Reference parameters



Reference [V] 9.0

☑ Automatic Adjust

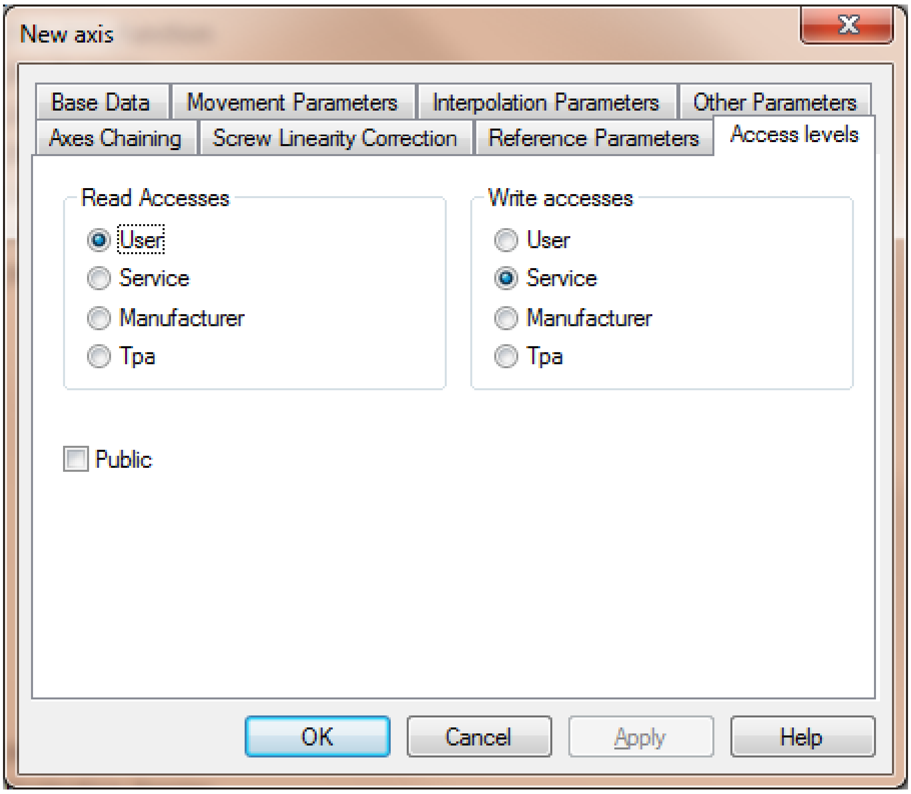
Initial Offset [V] 0

Notch Filter frequency [Hz] 0

		Negative	Positive
Minimum Voltage	[V]	0	0
Threshold	[V]	0	0

- **Reference:** value of the reference power corresponding to maximum speed
- **Automatic Adjust:** enables or disables calculation of automatic offset recovery. It's usually enabled.
- **Initial Offset:** Value to which initial reference offset is set. Value must be in the range -10 to 10. Default value is 0.
- **Notch filter frequency:** Frequency value to be filtered. Value must be in the range 0 to 500.
- **Minimum voltage:** Sets the minimum voltage parameters for the axis indicated. The negative value must be in the range -10 to 0, the positive value in the range 0 to +10. See SETDEADBAND instruction.
- **Threshold:** Sets the threshold values. They are always less than or equal to the respective minimum voltage values, hence the negative threshold value must be between 0 and the negative minimum voltage value. The maximum threshold value must be between 0 and the positive minimum voltage value.

Access levels



Read Accesses

● User
○ Service
○ Manufacturer
○ Tpa

Write accesses

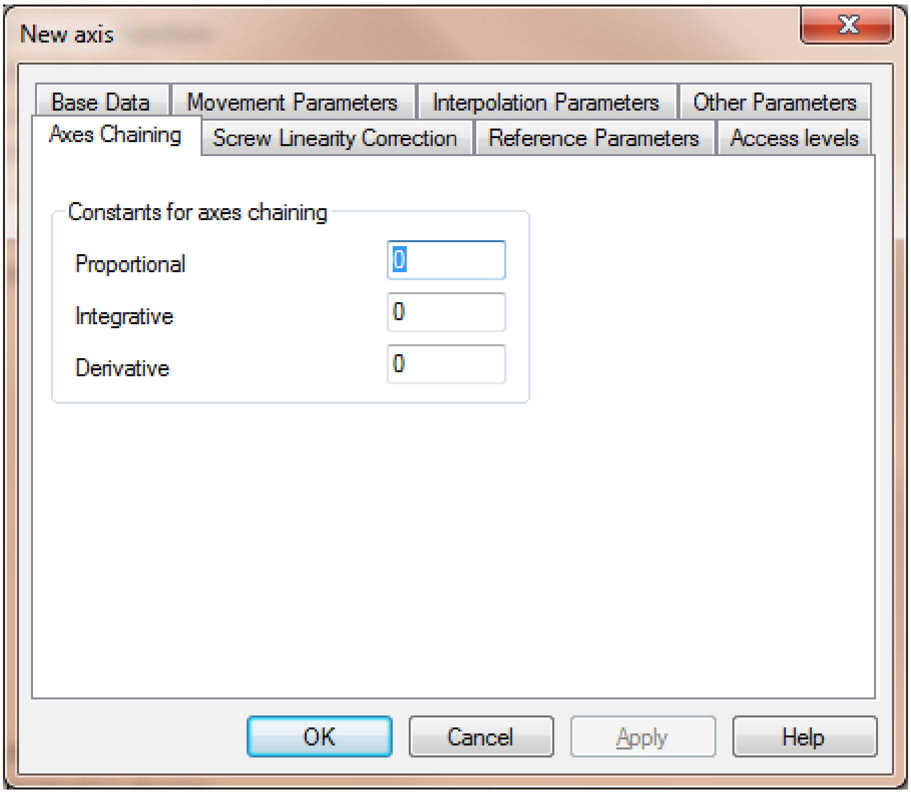
○ User
● Service
○ Manufacturer
○ Tpa

□ Public

- **Read Accesses:** specifying the minimum access level required for the axis to be visualised in the Diagnostic windows or in the Synoptic Views.
- **Write accesses:** specifying the minimum access level required to modify the state of the axis.
- **Public:** specifying whether the state of the axis can be read or modified by a GPL code not belonging to the group of the axis.

Axes chaining

Axes chaining parameters. These are the PID controller coefficients which correct the loop error difference between the master axis and the slave axes.



Constants for axes chaining

Proportional 0

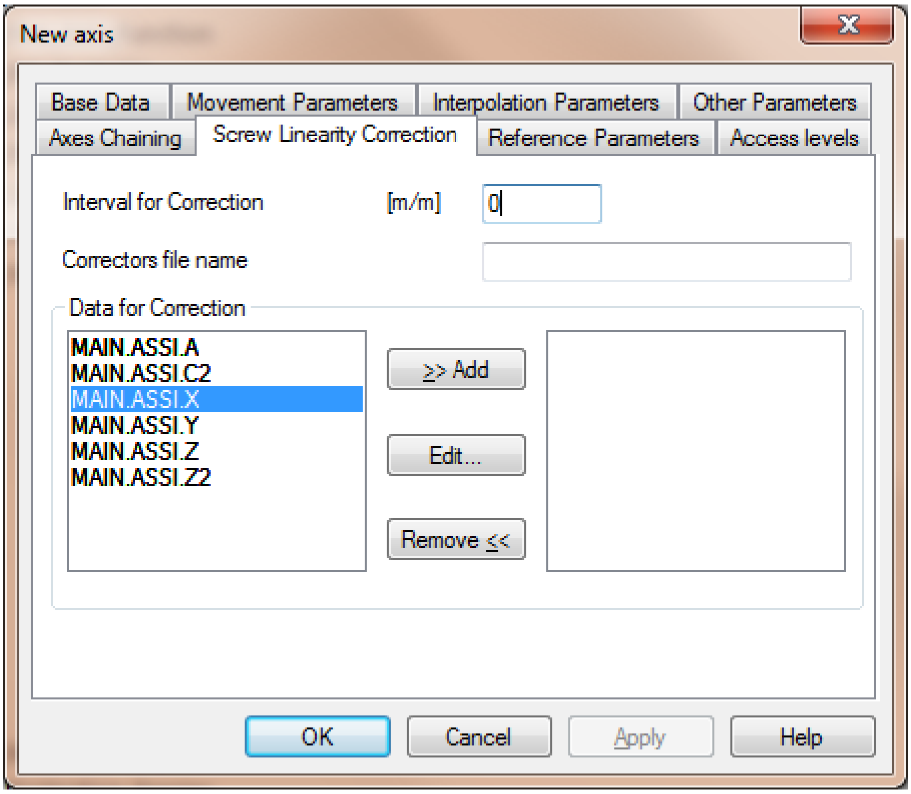
Integrative 0

Derivative 0

- **Proportional:** proportional coefficient
- **Integrative:** integration coefficient
- **Derivative:** derivation coefficient

Screw linearity correction

Setting the screw linearity correction of the axis. The correctors allow axis positioning errors to be compensated where these are due to mechanical imprecision of the axis itself (auto-correctors) as well as errors due to the effect deriving from the other axes of the machine (crossed correctors) typically related to bending in the structure. The correctors are not automatically enabled but must be enabled in the editing window for correction values (**[Edit...]** button) and activated with the GPL code using the command ENABLECORRECTION.



Interval for Correction [m/m] 0

Correctors file name

Data for Correction

MAIN.ASSI.A
MAIN.ASSI.C2
MAIN.ASSI.X
MAIN.ASSI.Y
MAIN.ASSI.Z
MAIN.ASSI.Z2

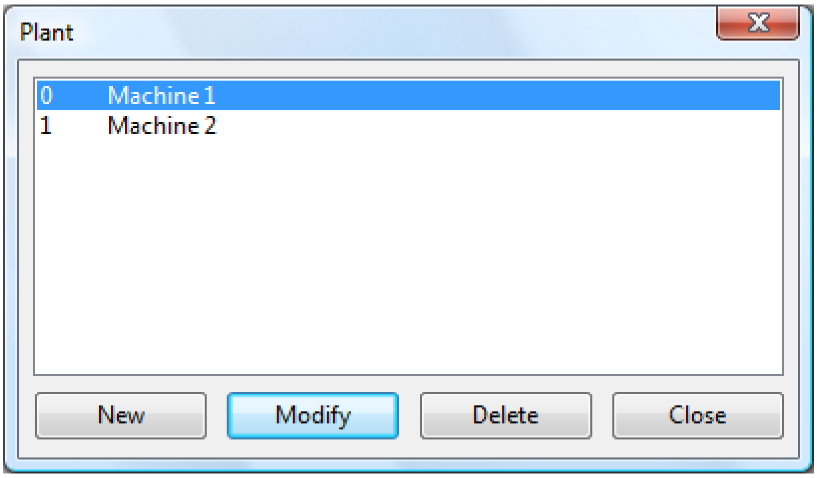
- **Correction interval:** this allows the distance between one correction and the next to be set. The measurement number is given by the length of the axis divided by the length of the correction interval.
- **Corrector file name:** this allows the name of the file in which the correction values are saved to be set. This will be an ASCII file in which the values are separated by the character ";". This allows them to be edited with a standard text editor. The file extension is not specified, the extension ".csv" (comma separated values) is automatically assigned.
- **Correction data:** allows the specification of the list of the axes to be included in the calculation of the correction of the current axis. The current axis is always included in the list, this means that the auto-corrector is always present. Up to another 5 axes can be specified. To add an axis select it in the list on the left and press the **[>>Add]** button. To remove an axis select it in the list on the right and press the **[Remove <<]** button. To specify correction values select an axis from the list on the right and press the **[Edit...]** button. A window is opened with a table in which to insert the correction values.

NOTE: There is a limit of ***235*** screw linearity corrections managed by the system for each axis. Consequently, the length of the measuring interval must be at least the 235th part of the length of the axis. For example, if an axis is 2500 mm long, the correction interval must be set at 10.63 mm or more.There is also a limit to the maximum value of an individual correction: this must be lower than 1024 encoder steps, for example for an axis with a resolution of 256 steps/mm the maximum correction is ±4 mm.

1.3 Logical Configuration

1.3.1 Plant configuration

To define a new machine or modify an existing one, access the Module Configuration screen page. Notice that, in this case, by Module Configuration we intend the configuration of the modules composing the plant, as confirmed by the heading of the configuration window shown in the following image: "Plant".



0	Machine 1
1	Machine 2

Plant configuration window

The Configuration environment can only be opened (from the manufacturer level or a higher level) when all the other work windows (synoptic, diagnostic, etc.) are closed, and it is done with the following command:

Access to Configuration

Select the heading ***Open Configuration*** from the ***File*** menu.

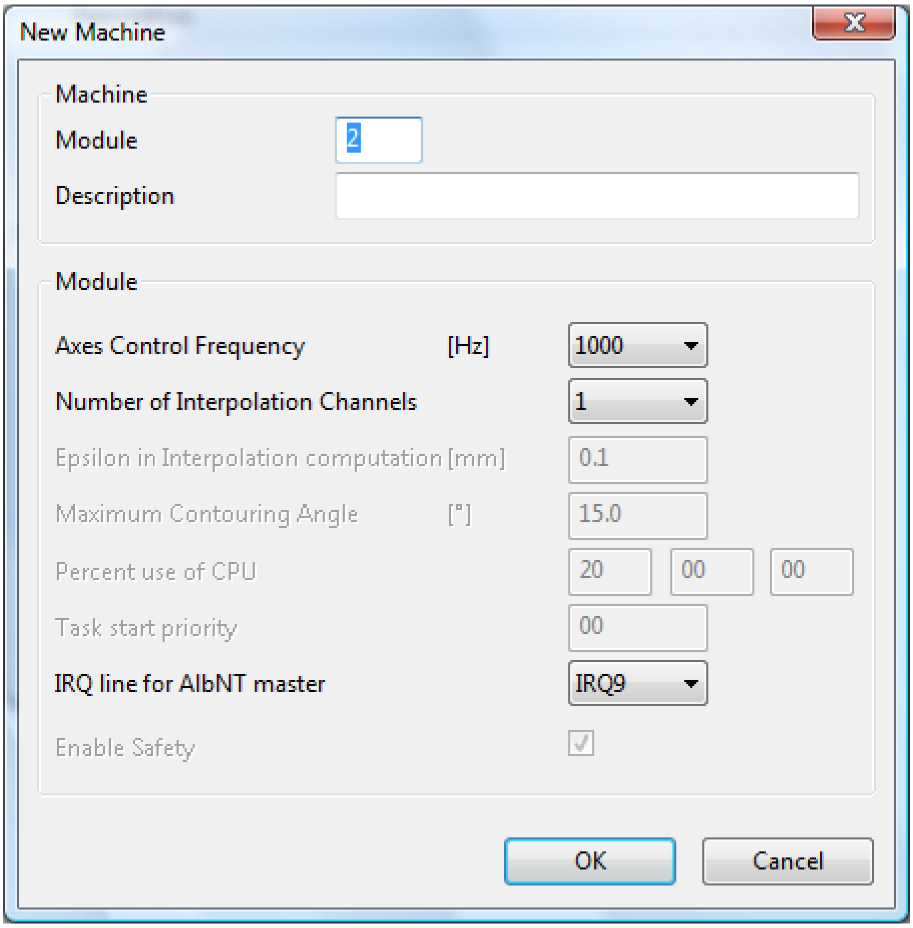
If no modules of the plant have been configured, Module Configuration is opened automatically, otherwise Machine Configuration will be opened. In this case, to access Module Configuration:

Select the heading ***Module Configuration*** from the ***Edit*** menu.

The window shown in fig. 8.1 will appear.

To add a module to the plant simply press **[New]**. **[Modify]** allows you to modify the data of an existing module, **[Delete]** to remove a module and **[Close]** to exit plant configuration.

When the **[New]** button is pressed, the window shown in the following figure is displayed.



New Machine

Machine

Module: 2

Description:

Module

Axes Control Frequency [Hz]: 1000

Number of Interpolation Channels: 1

Epsilon in Interpolation computation [mm]: 0.1

Maximum Contouring Angle [°]: 15.0

Percent use of CPU: 20 00 00

Task start priority: 00

IRQ line for AlbNT master: IRQ9

Enable Safety ☑

Module configuration window

The data that identifies the machine, to be specified, is:

- the number of the module: a progressive integer number which, if not specified, is assigned by the system
- a brief description

It also contains some data concerning the underlying Hardware. The same window can be opened from the branch of the Configuration Module for the groups and from the branch of the Hardware Configuration Module.

1.3.2 Groups Configuration

When the machine is designed from scratch it is necessary to define all the components and to write all the control cycles. However, we often develop projects for finished machines which have to be conveniently modified.
Because in the Albatros environment machines are organised following a hierarchic model (Machine, Group, Subgroup, Device), it is possible to create a file of loosely configured groups according to the most frequently used components. In this case, the machine can be designed by taking the required groups from the files and modifying them where necessary.

Therefore, the groups file is a collection of "standard" groups which allows you to design on a modular basis and, above all, to re-use the configuration work already done.

Creating a group

To create a new group access the Groups Configuration screen page. All the groups, sub-groups and devices come from the first branch of the tree, called Module. The Module branch cannot be modified. If you press the [ENTER] key or the [Modify] button, a dialog box opens to modify the module data.

Select the heading ***Groups*** from the ***Edit*** menu

From here it is possible to create new groups, to modify or delete existing ones and to copy a group giving the copy a new name.

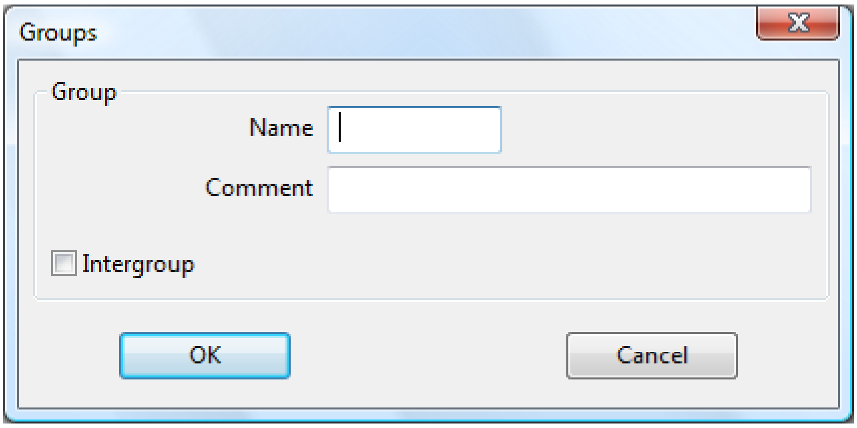
List of the commands to create, modify, delete groups, sub-groups and devices

Command	**Action**
Creating a new group, a sub-group, a device	[CTRL+ENTER], Button [New], Edit->New
Modifying a group, a sub-group, a device	[ENTER], Button [Modify],Edit->Modify
Deleting a group, a sub-group, a device	[DEL], Button [Delete],Edit->Delete

When you create a new group, the window below appears where following data must be set:

- the name of the group;
- a comment

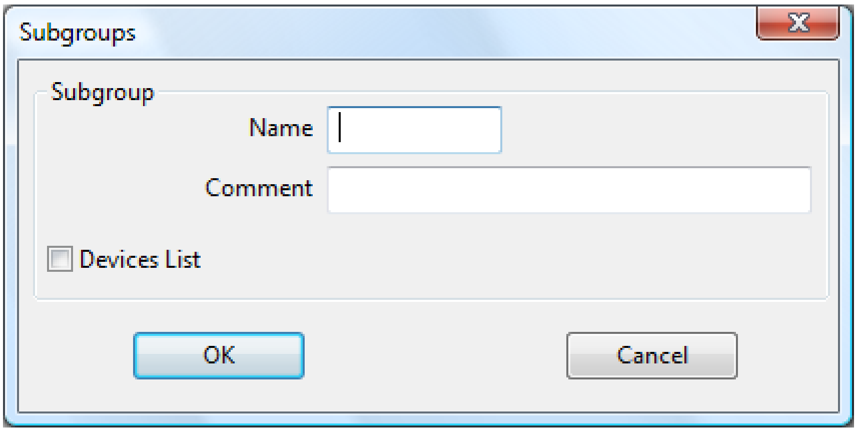
(both can be translated in the languages used by Albatros).



Group configuration window

It is also possible to indicate the group as **Intergroup**. This setting was used in previous versions of Albatros to allow the GPL code of other groups to access the devices and functions of this group. In this version the same result can be obtained by setting the devices as public. However at least one group must be set as Intergroup as this setting is used by Albatros to identify the "main" group of the machine. This is the group whose main function (the one with the same name as the group) is launched automatically when the machine is booted. The function of this mechanism is to start the machine and launch the tasks that verify that everything is functioning correctly, before passing the control to the user.

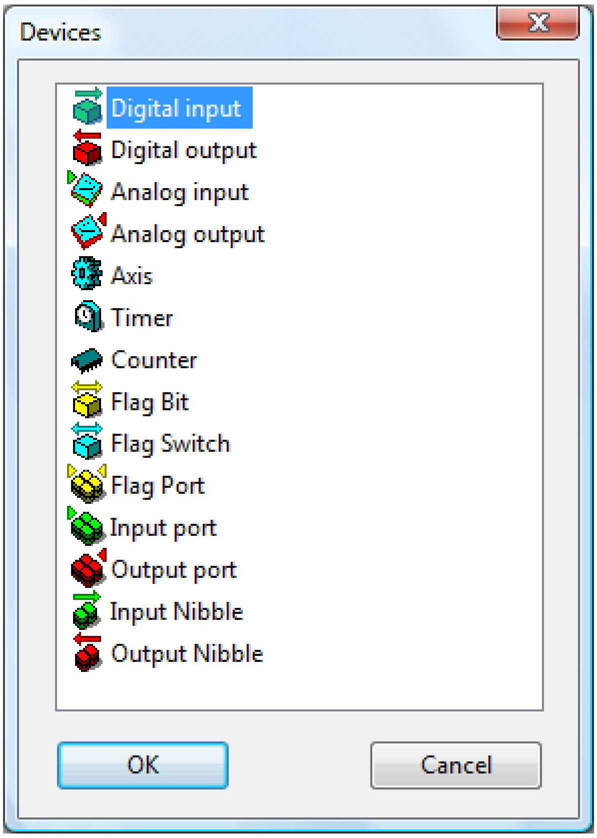
Adding devices to a group
To create a subgroup of the group, you must be positioned on the group. The window below opens.



Subgroup configuration window

If we do not intend to create any subgroups, select the *Devices* List, as in the figure below and press **[OK].** The name of the subgroup will be given automatically.

It is now possible to insert the single devices in the group. The process is similar to that used to create subgroups.
In this case a window containing the list of available devices will appear (see figure below).



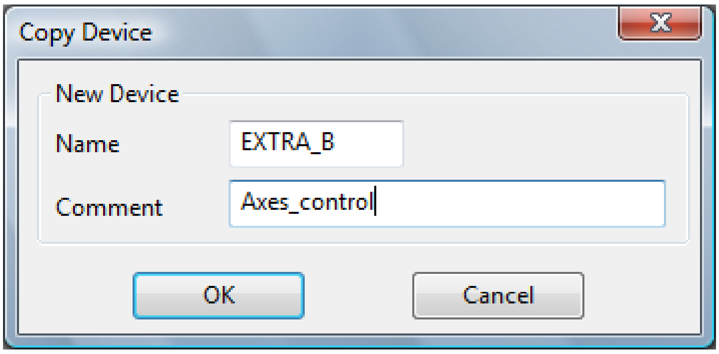
Devices selection window

Select the required device and press **[OK]** for confirmation.

Another window will appear, to enable us to enter a name, a comment and other data which varies according to the selected device. A detailed description of the devices and their settings will follow in the chapter Devices Configuration.

Copying a device

The device copy function allows you to make a copy of any device. First, select the device and then press **[Copy].** To insert the device in the list press **[Paste]** and enter the new name in the dialog window.



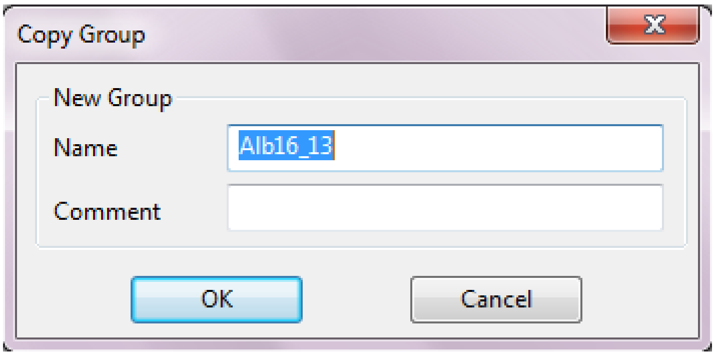
Name EXTRA_B

Comment Axes_control

Device copy window

Copying a group or subgroup

The group copy function allows you to copy a group with all the subgroups and devices it contains. Moreover, the corresponding group synoptic (having the same name as the group), if existent, is also copied.
This function allows you to create rapidly groups which have a similar structure to that of an existing group, without having to re-create all the devices one at a time. To copy a group, select the required group, press **[Copy]** and enter the new name of the group in the dialog window.



Name: Alb16_13

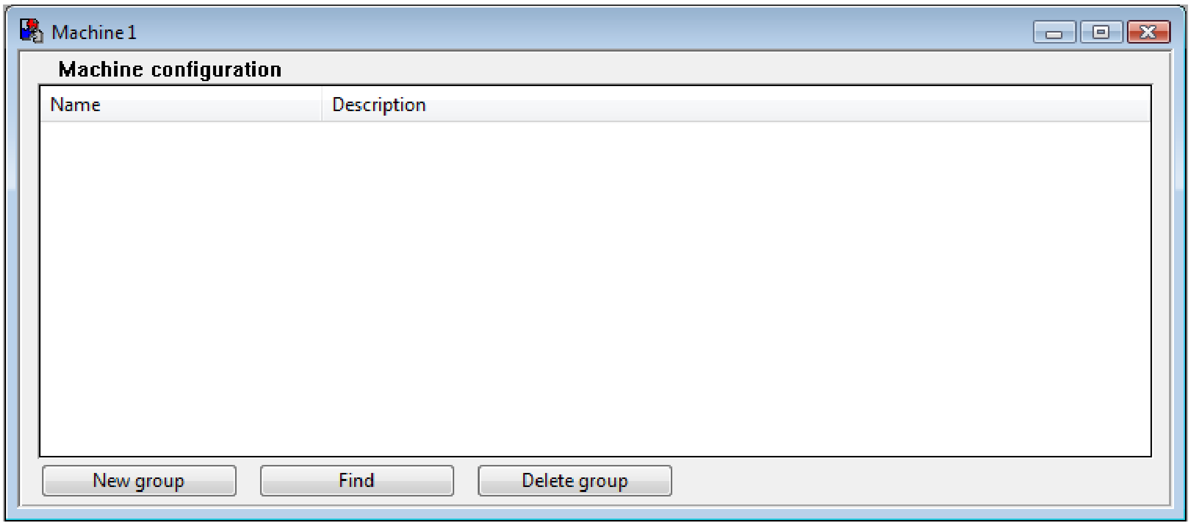
Group copy window

1.3.3 Machine configuration

Once all the groups and all the necessary devices within the groups have been configured, the Machine Configuration consists simply in selecting the groups which really exist .

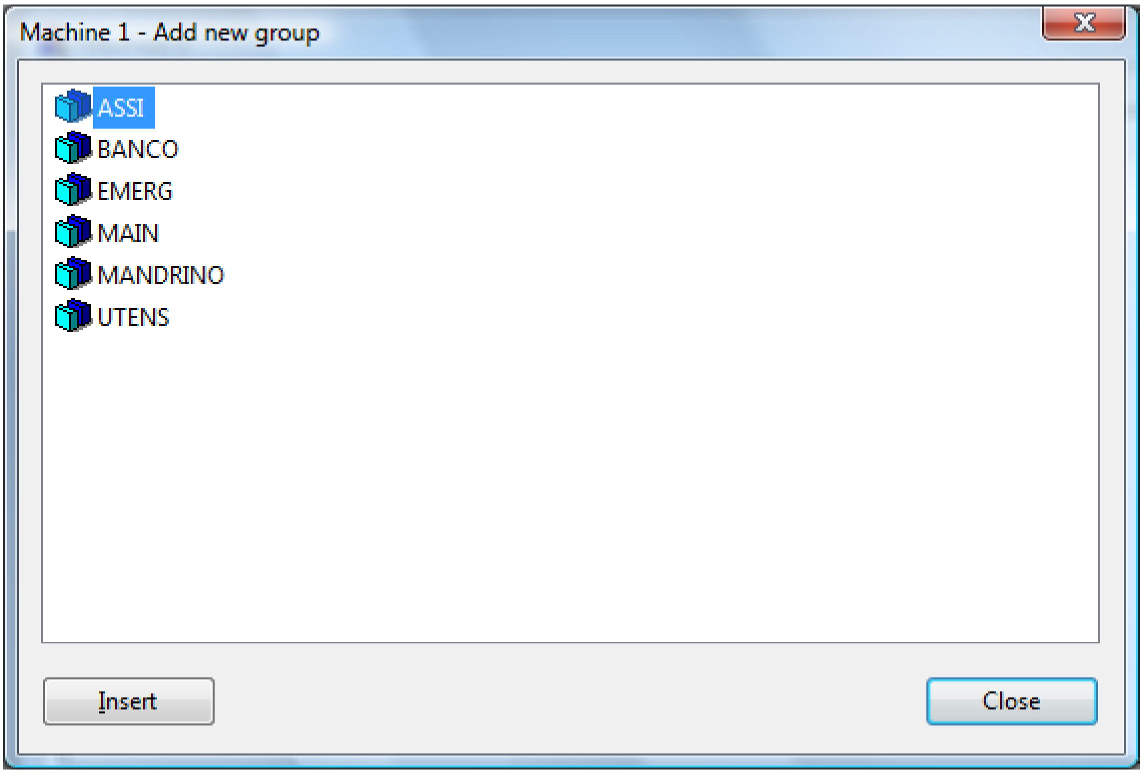
To access the Machine Configuration:

Select the heading ***Machine*** from the ***Edit*** menu.
The window shown in the following image will appear.



Machine Configuration

To insert a new group press **[New group]**. A window containing the list of all the groups contained in the file will appear.



Machine configuration: group selection window

At this stage, select the chosen group and **drag it with the mouse** to the Machine Configuration window. Notice that this is the only possible way of performing this operation.
It is also possible to remove an existing group or to search starting from the name of the group.

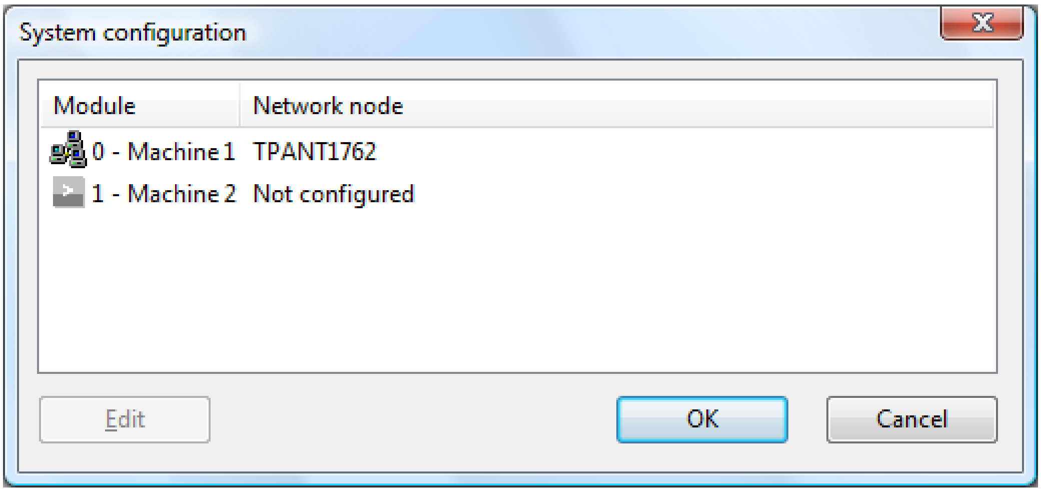
The machine can have only one intergroup. If more than one group was indicated as intergroup in the Group file, only one will have to be selected.

If necessary the configuration of certain devices may be modified, especially the axes configuration. In fact, it is possible to access the parameters of the devices through the machine configuration too, although most parameters can not be modified from this area. Remember also that any modification carried out in Machine configuration is not extended to the corresponding device in the group file.

1.4 Physical Configuration

1.4.1 System Configuration

The system configuration allows you to connect the physical resources (control units) to the modules defined in the logic configuration. This is possible into the System Configuration dialog box. The **modules** list of the plant is shown and to each of these a **Network Node**.



System configuration

Module	Network node
0 - Machine 1	TPANT1762
1 - Machine 2	Not configured

System Configuration

- **Local node** "Local" systems in which the HW handling the control is mounted directly on the user's system interface, that is the PC.
- **Name of a network node:** "Remote" systems in which the HW handling the control is connected to the PC through a serial line or network.
- **Not configured:** no configuration. This is the default at the beginning. If this choice remains, as a result it will be possible in the dialog box **Network Nodes Connections** to associate a remote module.

Up to 16 modules can be configured and one only can be configured as local node.
To assign a module, select the button **[Edit]** or double-click with the mouse on the network node to modify. Opening the pull-down menu, the list of the available remote modules is displayed, and it is also possible to use a local node or to set a module as not configured. To confirm the selection, select the button ✓.

N.B: The profile machining of Albatros is protected by a USB hardware key, configured by T.P.A. S.p.A.

1.4.2 Hardware Configuration

Hardware configuration consists in deciding what kind of board, plug or I/O remote units make up the system.
The card occupying at the first position is called Master board.
Albatros checks if the board of the hardware configuration is correctly inserted. The operator is informed of incongruences or of errors in inserting.
In this system boards can be configured on Can, POWERLINK II and EtherCAT-Buses.
In this case links between physical and logical devices are defined in an external file, whose name CANBUS.DEF is fixed for the Can-Bus, EPLBUS.DEF for PowerLinkII and ECATBUS.DEF for EtherCAT.
Clipper NT Embedded remote modules do not manage these field buses.

The TRS-AX , TRS-IO and TRS-16 remote modules can be connected only to TMSbus, TMSbus+ and TMSCombo+ boards.
No more than 4 TRS-AX remote modules can be connected to each TMSbus and TMSbus+ board.

Kinds of configurable cards:

• TMSbus boards	max. two
• TMSbus+	max. four
• TMSCombo+	max. four
• DualMech	max. four
• DualMech Mono	max. four
• TMSCan	max. two
• TMSCan+	max. four
• AlbNT	max. four
• AlbSLM board	max. four
• AlbMech	fino a due
• AlbIO32	max. two
• AlbNTPLC	one only
• CN2004	one only

These are the I/O remote modules that can be configured on GreenBus (v 3.0):

- Albre8 8 digital inputs and 8 digital outputs
- Albre16 16 channels which can be configured via software as digital input or output.
- Albre24 24 digital inputs and 24 digital outputs
- Albre48 48 digital inputs and 48 digital outputs
- Albrem 16 digital inputs and 16 digital output, 4 analog inputs and 4 analog outputs
- AlbSTEP 8 digital inputs and 6 digital outputs, one stepping motor
- AlbEV 20 or 24 electrovalves (D-sub 25 pin connector)
- AlbAPP keypad for manual movements and/or teach-in
- Albrea 4 analog input and 4 analog output

The configurable types of remote module on GreenBus /v.4.0) are as follows:

- TRS-AX 4 analog or step-by-step axes
- TRS-EV-24 24 electrovalves (D-sub 25 pin connector)
- TRS-16 16 channels which can be configured via software as digital input or output.
- TRS-IO 16 channels which can be configured via software as digital input or output. This can be expanded through TRS-IO-E and TRS-AN-E (max. 5 items) and TRS-AC-E modules.
- TRS-IO-E 16 channels which can be configured as digital input or output; they can only be used as expansion of a TRS-IO module.
- TRS-AN-E 1 analog input and 1 analog output that can be only used as an expansion of a TRS-IO module
- TRS-AC-E 1 counting axes and 2 digital inputs, configurable as zero position reference and fast input. In the table below the maximum number of TRS-AC-E, configurable in a TRS-IO

Number of TRS-IO-E and TRS-AN-E expansions	TRS-AC-E number expansion

The types of remote module that can be configured on EtherCAT are as follows:

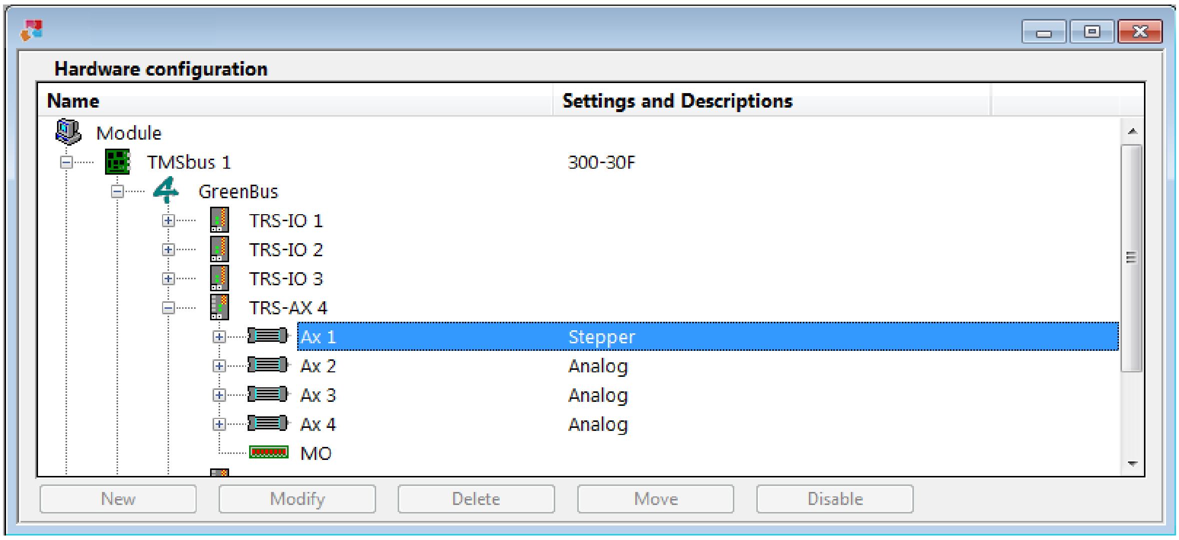
- TRS-CAT 16 channels that can be configured via software as a digital input or output. This can be expanded through TRS-IO-E and TRS-AN-E and TRS-AC-E modules.
- STAR-CATtransforms a EtherCAT network topology into a star topology by means of an input channel and up to 3 different output channels.

The table below shows the maximum number of expansions, that can be configured in a TRS-CAT.

Number of TRS-IO-E and TRS-AN-E expansions	TRS-AC-E expansion
7	0
5	1
3	2
1	3

Describing the hardware configuration window

The hardware configuration window opens if you select in the menu ***Edit->Hardware.***
To insert a board or a module of remote I/O, press **[New]**. In this way a window appear to select the board or the module of remote I/O and the position where it should be inserted.
In general, no more of 4 boards for each module and no more of 32 modules of remote I/O for each board can be configured . Hence, for each module you can configure up to 128 I/O modules. Regarding the TRS-AX remote modules a more precise clarification should be made; in fact, if the number of the inserted TRS-AX modules rises, the number of TRS-16 and TRS-IO, which can be used, decreases. To calculate the maximum number of TRS-16 and TRS-IO remotes, which can be inserted, you need to apply the following formula: number of other remotes = 32-(number of TRS-AX*4). For instance, if 3 TRS-AX are connected to a TMSbus, applying the formula we get: number of other remotes=32-(3*4), then no more than TRS-16 and/or TRS-IO 20 remote modules can be inserted.
The position of the remote in the list should be chosen according to the address set through a switch on the remote module. Please, make reference to the hardware documentation of the single remote.



Hardware configuration

Name	Settings and Descriptions
Module	
TMSbus 1	300-30F
GreenBus	
TRS-IO 1	
TRS-IO 2	
TRS-IO 3	
TRS-AX 4	
Ax 1	Stepper
Ax 2	Analog
Ax 3	Analog
Ax 4	Analog
MO	

Hardware Configuration

According to the selected board, it could be necessary to set the kind of axes managed. This is invalid for AlbSLM boards, AlbESlm expansions and remote TRS-AX, CN2004 board.
What kinds of axes can be associated to the various hardwares are described as follows:

• AlbNT board	analog axes and counting axes
• AlbENt expansion	analog axes and counting axes
• AlbSLM board	digital axes, frequence/direction axes (IV axis only, if configured as Frequence/Directon type, counting axes (Aux connectors only)
• AlbESlm expansion	digital axes, frequence/direction axes (IV axis only, if configured as Frequence/Directon type, counting axes (Aux connectors only)
• AlbMech board	digital axes
• DualMech board	digital axes
• DualMech Mono board	digital axes
• TRS-AX axes	analog axes (if configured as analog type), counting axes (if configured as analog type), step-by-step axes (if configured as Step-by-Step type)
• remote AlbStep	step-by-step axes
• TRS-AC-E expansion	counting axes

In the Mechatrolink II the number of the axes that can be configured changes according to the set value of the control frequency of the axes:

Board	**Axis frequency control (Hz)**	**Maximum number of servo drives**
AlbMech	1000	8
AlbMech	<=500	16
DualMech Mono	1000	8
DualMech Mono	500	20
DualMech Mono	250	30
DualMech	1000	16
DualMech	500	40
DualMech	250	60

The column **Settings and descriptions** shows or assigns some informations concerning the board or the remote module set.

Using **[Move]** you can move a board from a slot to another or a remote module from the bus of a board to the bus of another board. Through this operation any possible connections concerning the remote and available in the Virtual-Physical configuration are maintained. If the board to be moved contains some nodes configured on an external bus, this board cannot be moved. The same command can be selected from the ***Edit->Move*** command*.*

A remote module can also be disabled. Disabling has the effect of keeping the connections in the Virtual-Physical configuration whilst the remote module and the devices connected to it are totally disregarded by the system. Therefore, no error is generated if the module is not detected during initialisation and no error is generated when a GPL instruction is executed on a device associated with the module. *Consequently, this feature must be used with a special care*. To disable a remote module, use the **[Disable]** button; to enable a remote module again, use the **[Enable] button**. The same command can be selected from the ***Edit->Disabled*** command.

How to write CANBUS.DEF. file

Albatros can manage bus devices on CAN field bus through Tpa boards equipped with a CAN Bus connector or through generic boards for CANbus control. Connections between physical and logical devices on CANbus are defined in the CANBUS.DEF file, stored in the configuration folder of the corresponding module(\MODn\CONFIG). The formalism used is in accordance with the standard IEC1131.

Following description must exclusively be used with **TMSbus, TMSbus+ and TMSCan+ and TMSCan** boards. The main elements to define the CAN hardware are as follows:

CANBUS.DEF file is a text format file that describes the connections between logical devices and physical devices on Powerlink. For every module a EPLBUS.DEF file must be written and memorized into the configuration folder of corresponding module. (\MODn\CONFIG). Inside the file the part describing Powerlink hardware should come before the description of the logical-physical connections. The main elements to define the Powerlink hardware are follows:

- **(*…*)** beginning and ending of a comment. Comments can be written on more than one text line. You can enter a comment inside another. This is useful when you want to comment a block of definitions whose rows are commented. For example

```
.....
CN(3)  ID=17 IO RPDO=4 TPDO=8; (*one only RPDO and one only TPDO*)
(*
CN(4) ID=21 IO RPDO=2+2+3 TPDO=8; (*(* two RPDO1 of 2 bytes ....*)
CN(5) ID=22 IO RPDO=1+4 TPDO=8+8; (*two RPDOs and two TPDOs *)
*)
```

- **MN (number) attributes** beginning of the description block of a MN (managing node), that is master board of the *CanAddress*. Instead of a number you can use an alphanumeric identifier that will be used later on to identify MN in the description bloc of the logic-physical connections. In this case the attribute **ID** is obligatory. The number in brackets is the index, that will be used for the composition of the Can address. MN is configured by means of the following attributes:
  ID=index number of the board in the Albatros hardware configuration (from 1 onwards); if absent, MN(number) is used.
  TIME=number of sampling rate in msec. It cannot exceed 60000 (60 seconds)
  BAUDRATE=number of CAN communication rate in kilobits per second (can be 1000, 500, 250, 125, 100)
  TIMEPDO=time in msec. It shows the time devoted to the synchronous communication of the PDOs. The value set cannot exceed the TIME value (it not an obligatory value).
  Service name=YES (to enable the service), NO (to disable the service). It sets the CAN service or protocol, that can be enabled or disabled. Service list:

Service name	Description
SERVICE-EMCY	enables or disables the EMCY service
SERVICE-NMT	enables or disables the NMT service. If there are TMSCan and TMSCan+ boards, this service is always enabled.
SERVICE-CTRL	enables or disables the NoteGuarding and/or HeartBeat protocol check
SERVICE-SYNC	enables or disables the SYNC service
SERVICE-SDO	enables or disables the SDO service
SERVICE-PDO	enables or disables the PDO service
SERVICE-NGUARD	enables or disables the NGUARD service
SERVICE-RCOVER	enables or disables the RCOVER service: this service cannot be used, if there are TMSCan

	and TMSCan+ boards.
SERVICE-HBEAT	enables or disables the Heartbeat service for the nodes. This service cannot be used, if there are TMSbus and TMSBus+ boards.

Example: SERVICE-EMCY=YES (enabling the EMCY service). SERVICE-EMCY=YES (disabling the EMCY service).
TIMEAFTERRESET=time in msec. It shows the waiting time during the initial phase after a software reset of the nodes in the network. It cannot exceed 60000 (60 seconds).
LIFETIMEFACTOR=number. This is the number of CAN cycles without answer to the Node Guarding call before the generation of Disconnected node error. It cannot exceed 100 or be less than 1. (Default value: 3)

- **CN (number) attribute** beginning of description block of a CN (Controlled Node). The number in brackets is the index, that will be used for the composition of the *CanAddress*. Instead of a number you can use an alphanumeric identifier that will be used later on to identify CN in the description block of the logic-physical connections. In this case the attribute **ID** is obligatory. CN will be considered as a part of the CAN subnet of the previous MN description block. A CN is configured by means of the following attributes:
  IO indicates that it implements the DS401 (I/O) specification
  SERVO reserved
  DISABLED: disables CN. This word can be entered in any part of the definition, after CN() at the beginning and before ';' at the end.
  ID=number is the CN number (from 1 onward); if this field does not exist, CN(number) is used.
  RDPO=**list**: sequence of values (max. 8 for TMSBus and TMSBus+ boards, max. 4 for TMSCan and TMSCan+boards) , separated by the character '+'; each value identifies the dimension of a receiving or transmitting PDO (for TPDO) of CN (1÷8).M
  With TmsBus and TMSBus+ boards ror each PDO the COB-ID can be defined, enclosed within round brackets (Ex.: "RPDO=2+4+4+2+1(101)+4(102)").
  With TMSCan and TMSCan+ it is also possible to configure asynchronous PDOs, i.e. PDOs that are not updated at each cycle, but only on specific request. We define an asynchronous PDO by adding ASYNC . Asynchronous PDOs should be sent in the GPL code by means the SENDPDO instructions.
  TPDO= list: list of sequence of values (max. 8 TMSBus and TMSBus+ boards, max. 3 for TMSCan and TMSCan+ boards), separated by the character '+'; each value identifies the dimension of a receiving or transmitting PDO (for TPDO) of CN (1÷8).
  With TMSBus and TMSBus+ boards it is possible for each PDO to define the COB-ID, enclosed within round brackets (Ex.: "RPDO=2+4+4+2+1(101)+4(102)").
  With TMSCan and TMSCan+ it is possible to configure asynchronous PDOs, i.e. PDOs that are not updated at each cycle. We define an asynchronous PDO by adding ASYNC. Asynchronous PDOs should be received in the GPL code by means the RECEIVEPDO instructions.
  AUTOOP: this device allows the automatic passage to the Operational status after a reconnection (optional).
- ; ending a MN or CN description block

Following description, concerning the logical-physical connections, must be used for all the boards on CANBus

- **(*...*)** beginning and ending of a comment. Comments can be written on more than one text line. You can enter a comment inside another. This is useful when you want to comment a block of definitions whose rows are commented.
- **VAR** beginning of block of connections description
- **DeviceName** complete logical device name. It can be written with the form "Group.Subgroup.Device" or "Group.Device
- **AS** keyword that separates **DeviceName** from **CanAddress**
- **CanAddress** (for Tpa boards with CAN control) physical address on CANBus. The formalism for the description is:
  % is the first obligatory character.
  I or **Q** is the second character. **I** indicates an input device, **Q** indicates an output device
  X or **B** is the third character. **X** indicates that the following value must interpreted as bit, **B** indicates that the following value must be interpreted as byte. If omitted, the next value is interpreted as bit. Next characters are a sequence of numbers indicating the address. If in the system is configurated more than one Can board then it's possible to distinguish the board number by putting before the address the

number of the board followed by a point. The address can be expressed in base 2, 8, or 16 according to the formalism IEC.

- **CanAddress** (for TMSbus boards) shows the address at the beginning, how many bits in dualport and of which board are available. The formalism for the description is:
  % is the first obligatory character.
  I or **Q** is the second character, **I** showing an input device, **Q** shows an output device
  X o **B** is the third character. **X** shows that the next number must be interpreted as a bit, **B** shows that the next number must be interpreted as a byte. If omitted, the next value is interpreted as a bit.
  The next characters are a sequence of numbers indicating the address. They are separated by a dot. The first number refers to the master board (TMSBus) of the bus, the second one to the node, the third optional one is an offset within the node (this number is a progressive one depending on the Albatros device type. This offset can also be expressed in base 2, 8 or 16 according to the IEC formalism.
  If the offset is not available, we consider 0.
- **;** completes the description of a connection
- **END_VAR** ending of block of connections description

Whatever is found after the keyword END_VAR of block end is ignored.
Whatever is out of the blocks is ignored.
The correctness of file whether from the point of view of the syntax, or from the point of view of the contents is verified during Albatros starting. In case of errors notice, is visualized an error message. The description of all errors is in file ERRCAN.TXT memorized into the folder defined in Tpa.ini at option DirReport.

Example of definition of CAN Hardware on Tpa board:

```
MN(1)      TIME=10 BAUDRATE=1000;
CN(3)      ID=17 IO RPDO=4 TPDO=8;      (* one only RPDO and one only TPDO *)
CN(4)      ID=21 IO RPDO=2+2+3          (* two RPDO1 with 2 bytes and a RPDO3 with 3
           TPDO=8;                      bytes *)
CN(5)      ID=22 IO RPDO=1+4 TPDO=8     (* two RPDO and two TPDO *)
           +8;

VAR
    Main.EV1              AS %QX1.30.10;
    Main.EV2              AS %Q1.3.11;
    Main.Assi.InpPort     AS %B1.5.12;
    Emerg.InputW          AS %IX2.5.13;        (*board 2 *)

END_VAR
```

Example of definition of CAN Hardware on generic boards:

```
VAR
     Main.EV1              AS %QX10;          (* output device 10  board 1*)
     Main.EV2              AS %Q11;           (* output device bit 11 board 1*)
     Main.Assi.InpPort     AS %B12;           (* input device byte 12 board 1*)
     Emerg.InputW          AS %IX2.13;        (* input device bit 13 board 2*)

 END_VAR
```

How to write CANBUS.DEF file for S-CAN devices per dispositivi S-CAN

The description of the S-CAN hardware configuration is defined in the CANBUS.DEF text file, stored in the configuration folder of the corresponding module. (\MODn\CONFIG). The formalism used is in accordance with the standard IEC1131.

Following description must be exclusively used with **TMSbus**, **TMSbus+** boards. The main elements to define the S-CAN hardware are as follows:

- **(*…*)** Beginning and end of a comment. Comments can be written on more than one text line.
  You can enter a comment inside another. This is useful when you want to comment a block of definitions whose rows are commented. For example
  .....
  CN(3) SERVO RPDO=8 TPDO=8; (*servo*)

```
(*
CN(4) SERVO RPDO=8 TPDO=8; (*servo ....*)
CN(5) SERVO RPDO=8 TPDO=8; (*servo...*)
*)
```

- **MN(*number*) attributes** beginning of the description block of a MN (managing node), that is master board of the S-CAN communication. The number in brackets is the index that will be used for the composition of *CanAddress*. Instead of a number you can use an alphanumeric identifier that will be used later on to identify MN in the description bloc of the logic-physical connections. In this case the attribute **ID** is obligatory. A MN is configured by means of the following attributes:
  S-CAN shows the type of CAN protocol. It is obligatory.
  ID=*index number* of the board in Albatros hardware configuration (from 1 onwards); if not present, MN(*number*) is used
  TIME=*number* sampling time in msec (accepted values 2,4 and 6 only).
  BAUDRATE=*number* CAN communication speed in kilobits/second (it can be 1000, 500, 250, 125, 100)
  TIMEAFTERRESET=time in msec. It shows the waiting time during the initial phase after a software reset of the nodes in the network. It cannot exceed 60000 (60 seconds).
  LIFETIMEFACTOR=number. This is the number of CAN cycles without answer to the Node Guarding call before the generation of Disconnected node error. It cannot exceed 100 or be less than 1. (Default value: 3)
- **CN(*number*) attributes** beginning of description block of a CN (Controlled Node). The number in brackets is the index that will be used for the composition of *CanAddress*. Instead of a number you can use an alphanumeric identifier that will be used later on to identify CN in the description block of the logic-physical connections. In this case the attribute **ID** is obligatory. CN will be considered as a part of the S-CAN subnet of the previous MN description block. A CN is configured by means of the following attributes:
  SERVO shows that it is a servo drive. It is obligatory.
  DISABLED: disables CN. This word can be entered in any part of the definition, after CN() at the beginning and before ';' at the end.
  ID=*number* is the CN number (from 1 onward); if there is not this field, CN(number) is used
  RDPO= *sequence list* of values (max. 8) separated by the '+' character, each one identifying the dimension of a CN reception PDO (1÷8).). For each PDO the COB-ID can be defined, enclosed within round brackets (Ex.: "RPDO=2+4+4+2 +1(101)+4(102)").
  TPDO=*sequence list* of values (max. 8) separated by the '+' character, each one identifying the dimension of a CN reception PDO (1÷8).). For each PDO the COB-ID can be defined, enclosed within round brackets.
- **;** ending a MN or CN description block

Following description defines the logical-physical connections

- **(*...*)** Beginning and end of a comment. Comments can be written on more than one text line. You can enter a comment inside another. This is useful when you want to comment a block of definitions whose rows are commented.
- **VAR** beginning of block of connections description
- ***DeviceName*** complete name of the logical device. It can be written under the "Group.Subgroup.Device" or "Group.Device" form.
- **AS** Keyword separating ***DeviceName*** from ***CanAddress***
- ***CanAddress*** shows the address at the beginning, how many bits in dualport and of which board are available. The formalism for the description is:
  % is the first character, it is obligatory.
  I or **Q** is the second character. **I** shows an input device, **Q** shows an output device
  X or **B** or **L** is the third character. **X** shows that the following value has to be interpreted as bit (digital inputs and outputs), **B** shows that the next value has to be interpreted as byte, **L** shows that the next value has to be interpreted as 8 bytes (axes). If omitted, the next value is interpreted as bit.
  The following characters are a set of figures, separated by a point '**.**', identifying the address The first number refers to the MN (TMSbus) of the bus, the second one to the CN, the third one, optional, is an "offset" inside the CN (such a number is a progressive one linked to the Albatros device typology); the offset can also be expressed with base 2, 8, or 16 according to the IEC formalism.
  If the offset is not available, we consider 0. In the S-CAN drive the offset can be used to send some commands to the drive by means of analog outputs. The following table provides the commands that can be sent and the offset address. The first three are digital outputs and the last one is a output port.

Command	**Offset**	**Example**

Servo on	0	Ax.ServoOnX AS %QX1.1.0;
Enabling movement	1	Ax.EnableX AS %QX1.1.1;
Stop in the ramp	2	Ax.StopX AS %QX1.1.2;
Reset alarms	3	Ax.ResAlmX AS %QX1.1.3;
Sending couple value	8	Ax.TorqueX AS %QB1.1.8;

- **;** completes the description of a connection
- **END_VAR** end of the description block of the connections

Whatever is found after the keyword END_VAR of block end is ignored.
At Albatros startup the program checks if the file is correct from bot the points of view of the syntax and of the content. If errors are detected, an error message is displayed. Each error description is provided in the file ERRCAN.TXT stored into the folder defined in Tpa.ini under DirReport.

Example of definition of S-CAN Hardware on TMSbus board:

```
 MN(1)                         S-CAN TIME=2 BAUDRATE=1000;
    CN(1)                      SERVO RPDO=8 TPDO=8;
    CN(2)                      SERVO RPDO=8 TPDO=8;


VAR
       Ax.X                    AS %IL1.1
       Ax.Y                    AS %IL1.2
END_VAR
```

Characteristics of the EtherCat Management in Albatros

The communication mode is always DC-Synchronous. The first node of the network provides the clock, so it is essential to make sure that that node provides a precise and stable clock, as it is provided for example by TRS-CAT. It is not possible to use other modes, such as, for example, Free-Run. Managed protocols are: CoE (CAN application protocol over EtherCAT) and EoE (Ethernet over EtherCAT). Inside CoE, the device profiles DS401 and DS402 are managed by the default operating mode of the axis control *cyclic synchronous speed mode*.

The maximum number of EtherCat nodes is 200.

Foreword

To each physical EtherCAT device an ESI file(EtherCAT Slave Information) is associated, describing the characteristics and the functionalities of the device. . This file is in XML format. For each device one only ESI file must exist. Generally, the ESI files can be downloaded from the manufacturer's Internet site. Albatros searches for these files in the folder defined in Tpa.ini in the section [tpa] under DirESIFiles. Default option is the subfolder ETHERCAT of SYSTEM. "\EtherCAT" di SYSTEM.

From the ESI Albatros files it obtains the information on the device, by analysing all the elements "/ Devices/Device/Type". Each device is identified by a Vendor ID, a Product ID and by a Revision Number. If more than a device with the same name is available, the same Vendor ID and the same Product ID, that with the greatest Revision Number is considered.

Always from the ESI files the information on the expansions (also called modules) of the devices are obtained. Albatros finds the information on the types of expansions by searching in the ESI file of the device the elements "Modules/Module".

ECATBUS.DEF file

ECATBUS.DEF file is a text format file that describes the hardware configuration and the connections between logical devices and physical devices on EtherCAT. In each module using this bus a ECATBUS.DEF file must be written and stored in the configuration folder of the corresponding module (\MOD.n\CONFIG).

The file is divided into two sections, the first is the one that describes the hardware EtherCAT, and is equivalent to the Hardware Configuration of Albatros. In this section the physical devices are listed, that is the nodes of the EtherCAT network and their settings. The second section corresponds to the Virtual-Physical Configuration; in this section the couplings between logical devices and single inputs and physical outputs of the different EtherCAT nodes are listed. This section of the files is enclosed between

VAR and END_VAR key words.

Each single definition available in the file in the hardware configuration section or in that of the Virtual-Physical can be described on more rows and must be finished by the ‘;’. character. To enter some comments (or bypass part of virtual-physical and hardware configuration) you enter the characters ' (* ' at the beginning of the comment text and the ' * characters) ' at the end. The comments can be on multiple lines of text. You can also enter a comment inside another one. This is useful when you want to comment a block of definitions whose rows are commented.

Example:

```
(* Beginning of the EtherCAT configuration of the module *)
(*
    Here you must enter the definitions concerning EtherCAT hardware
*)
VAR
(*
    Here you must enter the virtual-physical associations among Albatros and I/O EtherCAT logical
   devices.
   (* This is a comment inside another comment*)
*)
END_VAR
```

EtherCAT Hardware configuration

The hardware is configured by describing the master boards and, for each board, the list of physical devices connected to that card on the bus. The physical devices are also called "nodes" of the field bus. For EtherCAT the master board is not a specific board of bus control, but a network connection of the module is used. As for the local modules the network connection must be one of those managed by RTX, while for the remote modules a specific network connection of the module is used among those managed by Windows CE 6.0. For each local or remote module, you can configure one master only.

The master board is identified in the ECATBUS.DEF master file as MN, i.e Managing Node, while each hardware device or node, is identified as CN, i.e Controlled Node.

The syntax to describe the master (MN) is the following (please, note that the definition is finished by the

‘;’) character:

```
MN(index)                    (* index is the number to use for the virtual-physical* connections)
   ID=address                (* board number, from 1 on; optional *)
   NAME=interface_name       (* name of the network interface*)
;
```

Where:

index
can be a number, from 1 onwards or an alphanumeric identifier. It will be used in the virtual-physical section (i.e. between VAR and END_VAR) to show the master board in the EtherCAT network of which the node to be associated to the logical device is placed. If you use an alphanumeric identifier, in the definition of MN() you must also specify the ID=address.

address
Board number associated to the EtherCAT bus managed by this MN(). It must be a number from 1 onwards, if the index field is not used. If the index is an alphanumeric identifier, then the address must be defined in an explicit way.

interface_name
this is the name of the interface acting as EtherCAT master. For the local modules the default value is “rtnd0”, that is the name of the section describing the network interface inside the ini file of RTX di IntervalZero. The default name of CN2008 remote module is "RTCENIC1", the default name of CN2128 remote module is "E1Q51CE61".

Example: MN(1) . Full example in the paragraph **"Example of EtherCAT hardware configuration"**.

The syntax to describe the node (CN) is the following (also in this case the definition is finished by this character ‘;’):

```
CN(index)          (* index is the number to use for the virtual-physical*)
ID=address         (* address of the node, from 1 on *)
TYPE=device_name   (* name of hardware device *)
RxPDO=pdo_sequen   (* description of a PDO that the node receive; optional *)
ce
TxPDO=pdo_sequenc  (* description of a PDO that the node sends; optional *)
e
OPMODE=axis_mode   (* servo nodes, operating mode of axis control; optional *)
DISABLED           (* disables the node; optional *)
 IO                (* considers the node as of I/O even if it is a servo; optional *)
;
```

Where:

index

can be a number, from 1 onwards or an alphanumeric identifier. It will be used in the virtual-physical section (i.e. between VAR and END_VAR) to show the node to be associated with the logical device. If you use an alphanumeric identifier, in the definition of CN() you must specify also the ID=address.

Examples:
CN(100) ID=+ TYPE=TRS-CAT:AN-E:IO-E;
CN(200) ID=+ TYPE=STAR-CAT;
 CN(101) ID=+ TYPE=TRS-CAT;
 CN(LTi_1) ID=+ TYPE=3-Axis-module;
 Full example in the paragraph **"Example of EtherCAT hardware configuration"**.

address

Node number of the EtherCAT bus. It must be a number from 1 onwards and if it is not indicated, the index field is used. If the index is an alphanumeric identifier, then the address must be defined in an explicit way.

device_name

Name of the device that is searched in the file ESI. This name can be indicated in several ways. It is worth using (1) the name that is in the tag Device\Type (even only a part of the name), but (2) it is accepted also that in the tag \Device\Name, or (3) you can write Product ID and the Vendor ID separated by a point (‘.’). Examples:
Examples:
TYPE=3-Axis-module
TYPE=i700_(Double
 Full example in the paragraph **"Example of EtherCAT hardware configuration"**.

The devices can also have some expansions (called also modules) are they also must be indicated, by making the name of the device follow the list of the expansions, separated by the ':' character. :'. the mandatory modules (“mandatory”) are automatically added and must not be indicated. if the device or the expansion name contain some space characters (‘ ’), these can be replaced by underscores (‘_’).

It is not necessary to write all the components of the device and the expansion name, but it is sufficient to write those necessary for the univocal identification of the device and the expansion among all the ESI files.
Example:
TYPE=i700_(Double.
. Full example in the paragraph **"Example of EtherCAT hardware configuration"**.

pdo_sequence

A PDO (Process Data Object) is a communication object defined by the communication parameter and by the mapped PDO objects (max.8). PDOs are transmitted in the form "without confirmation". (see paragraph Description of a PDO).

axis_mode

defines the operating mode to be used for the nodes of drive type, i.e. for the nodes adhering to DS402 (object 6060_{16}). The mode is one of the following:

HOMING	Homing
VELOCITY	Velocity
PROF-POSITION	Profile position
PROF-VELOCITY	Profile velocity
PROF-TORQUE	Profile torque
INTERPOLATED	Interpolated position
SYNC-POSITION	Cyclic synchronous position
SYNC-VELOCITY	Cyclic synchronous velocity
SYNC-TORQUE	Cyclic synchronous torque

If it is not set, SYNC-VELOCITY will be used. At the moment, this is the only mode supported in a native way by the numeric control.

It is possible to add some attributes to the node definition:

DISABLED

This attribute indicates that the node is not present on the bus. Its configuration is examined, but it is not sent to the numeric control by Albatros. The same result could be obtained by commenting the whole definition of CN(), but some errors could be reported by analysing the EtherCAT virtual-physical configuration. The use of this attribute makes possible for the the logical devices associated with the node to be considered as not connected. Additionally, in the Albatros hardware configuration window this node is available and marked as disabled.

Example:
CN(44) ID=+ TYPE=SGDV-E1 DISABLED
Full example in the paragraph **"Example of EtherCAT hardware configuration"**.

IO

It is sometimes useful to force the numeric control to consider a particular node of axes as if it were an I/O node. This attribute applies to nodes only that support DS402 (servodrives).

In the configuration file, the definition of the several CN () must follow the definition of the MN(), like in a tree structure, in which each leaf is fastened to a branch.

Description of a PDO

You can define up to eight PDOs sent by the node (TxPDO) and up to eight PDOs received by the node (RxPDO). Each RxPDO describes one only PDO that the node receives from the master, therefore digital and analog outputs for I/O nodes or target velocity and controlword for axis nodes. Each TxPDO describes one only PDO that the node sends to the master, therefore digital and analog inputs for I/O nodes or current position and statusword for axis nodes.

For the list and the description of the PDOs and of the objects that can be mapped on a PDO please, make reference to the documentation of the specific EtherCAT device and to its ESI file.

In the description of PDOs you can use the formalism IEC1131-3 to indicate the numbers, i.e. the sequence of figures representing the number with base 10. However, if it starts by “16#” so the number is considered to be base 16 and also the characters from A to F (case-insensitive) are considered. If it starts by “8#”, it is considered to be base 8 and the allowed characters range from 0 to 7. If it starts by “2#”, it is considered to be base 2, therefore only the figures 0 and 1 are allowed. In the figures you can enter the underscore ‘_’ character to improve the readibility.

Example:
TYPE=i700_(Double
Full example in the paragraph **"Example of EtherCAT hardware configuration"**.

There are three modes to describe the PDOs in a CN:

1. Do not set any PDO.
   In this way the numeric control uses PDO configured by default in the device. This is the easiest mode and fits the majority of the CNs.
   Example:
   CN(100) ID=+ TYPE=TRS-CAT:AN-E:IO-E; Full example in the paragraph **"Example of hardware EtherCAT configuration"**.

2. Set only the PDOs without providing any list of the objects.
   To be used when a CN has several alternatives and not programmable PDOs. To use it, write TxPDO or RxPDO to set the direction of the data, followed by the '=' character and then by the number of the communication object (PDO number), without spaces in the middle.
   Example:
   CN(EL3102_1) ID=+ TYPE=EL3102 TXPDO=16#1A10; Full example in the paragraph **"Example of hardware EtherCAT configuration"**.

3. Describe the PDO in a complete way, setting the communication object and the list of the objects to map.
   This mode is the one that provides the best control over the information sent and received by the CN. To use this mode, describe the PDO like for the previous mode, then add the ':' character and the list of the object to map, joined together by the '+' character.
   Example
   RXPDO=16#1600:16#6040+16#60FF+16#6060
   RXPDO=16#1610:16#6840+16#68FF+16#6860
   RXPDO=16#1620:16#7040+16#70FF+16#7060
   Full example in the paragraph **"Exemple of EtherCAT hardware configuration"**.

Each object is described by its index in the object dictionary of CN, optionally followed by a sub-index. If the sub-index is not available, it is considered as 0.
Example:
TXPDO=16#1A00:16#6041+16#6064+16#6061+16#2918.1+16#6077+16#606C
TXPDO=16#1A10:16#6841+16#6864+16#6861+16#3118.1+16#6877+16#686C
TXPDO=16#1A20:16#7041+16#7064+16#7061+16#3918.1+16#7077+16#706C;
Full example in the paragraph **"Example of EtherCAT hardware configuration"**.

The dictionary object (object dictionary) is the core of every device. It enables the access to all the types of the device data, to the communication parameters, to the configuration and data processing parameters.

Attention: not all the object of the object dictionary can be mapped in a PDO.

Examples of description of objects in the configuration file:

16#7060 (* index with base 10: 28768; sub-index: 0 *)

16#2918.1 (* index with base 10: 10520; sub-index: 1 *)

As for the CNs of servodrives there is a PDO for each drive, so that the nth TxPDO and the nth RxPDO of the CN make reference to the nth drive of the CN. The first two objects of each RxPDO and TxPDO have a preassigned significance and dimension, i.e.:

Drive	RxPDO		TxPDO	
	1° object 16 bit Controlword	2° object 32 bit Target velocity	1° object 16 bit Statusword	2° object 32 bit Actual position
1° drive	16#6040	16#60FF	16#6041	16#6064
2° drive	16#6840	16#68FF	16#6841	16#6864

nth drive	Add 16#800 to each object of the preceding drive.

When you need to describe a PDO completely, you can use some automatic features that will simplify the description:

- If the PDO number is missing the first programmable PDO among those listed in the ESI file of the device is used;
  Example:
  RXPDO=:+16#6060
  TXPDO=:+16#6077;
- in the case of servodrives you can replace the list of Controlword e Target velocity with the character '+'; idem for Statusword and Actual position.
  Example:
  RXPDO=:+16#6060
  TXPDO=:+16#6077;

Full example in the paragraph **"Example of EtherCAT hardware configuration"**.

Reading or writing objects can be added for a specific drive by adding the index (and any subindex) of each object in the PDO of the drive.
Example:
RXPDO=:+16#6060
Full example in the paragraph **"Example of EtherCAT hardware configuration"** .

Then, these values can be read by GPL through the GETAXIS instruction (see the related chapter). It also possible to trace the additional objects both from the calibration window and from the oscilloscope.

More generally, it is possible to access specific objects in reading and writing within PDO through the GETPDO and SETPDO instructions (see the related chapters)

Each object inserted in a PDO must be described also in the ESI file of the Ethercat device. If that is not the case, when Albatros reads the ECATBUS.DEF file reports as warning the use of an unknown object and presets the length of 32-bit object.

Example of EtherCAT hardware configuration

```
MN(1) NAME=RTND0;

  CN(100)  ID=+  TYPE=TRS-CAT:AN-E:IO-E;
  CN(200)  ID=+  TYPE=STAR-CAT;
  CN(101)  ID=+  TYPE=TRS-CAT;

  CN(LTi_1)  ID=+  TYPE=3-Axis-module;

  CN(LTi_3)  ID=+  TYPE=3-Axis-module
          RXPDO=16#1600:16#6040+16#60FF+16#6060
          RXPDO=16#1610:16#6840+16#68FF+16#6860
          RXPDO=16#1620:16#7040+16#70FF+16#7060
          TXPDO=16#1A00:16#6041+16#6064+16#6061+16#2918.1+16#6077+16#606C
          TXPDO=16#1A10:16#6841+16#6864+16#6861+16#3118.1+16#6877+16#686C
          TXPDO=16#1A20:16#7041+16#7064+16#7061+16#3918.1+16#7077+16#706C;

  CN(LTi_4)  ID=+  TYPE=1-Axis-module
          RXPDO=:+
          TXPDO=:+16#6077;

  CN(10)  ID=+  TYPE=i700_(Double
          RXPDO=16#1605:16#6040+16#60FF+16#6060
          TXPDO=16#1A05:16#6041+16#6064+16#6061+16#6077+16#606C;

  CN(11)  ID=+  TYPE=i700_(Double
          RXPDO=:+16#6060
          RXPDO=:+16#6860
          TXPDO=:+16#6061+16#6077+16#606C
          TXPDO=:+16#6861+16#6877+16#686C ;
```

```
 CN(20)       ID=+  TYPE=I/O-System:EPM-S202:EPM-S302;
 CN(102)      ID=+  TYPE=TRS-CAT:AN-E:IO-E;
 CN(EK1100)   ID=+  TYPE=EK1100;
 CN(EL3102_1) ID=+  TYPE=EL3102     TXPDO=16#1A10;
 CN(EL3102_2) ID=+  TYPE=EL3102     TXPDO=16#1A10;
 CN(EL4031)   ID=+  TYPE=EL4031;
 CN(EK1100)   ID=+  TYPE=EK1100;
 CN(40)       ID=+  TYPE=EL2809;
 CN(41)       ID=+  TYPE=EL1809;
 CN(42)       ID=+  TYPE=EK1122;
 CN(43)       ID=+  TYPE=L7NH
        (* RXPDO=16#1600:16#6040+16#60FF+16#6060
           RXPDO=16#1601:16#6040+16#60FF+16#6060
           RXPDO=16#1602:16#6040+16#60FF+16#6060
           RXPDO=16#1603:16#6040+16#60FF+16#6060 *)
        (* TXPDO=16#1A00:16#6041+16#6064+16#6061+16#6077+16#606C
           TXPDO=16#1A01:16#6041+16#6064+16#6061+16#6077+16#606C
           TXPDO=16#1A02:16#6041+16#6064+16#6061+16#6077+16#606C
           TXPDO=16#1A03:16#6041+16#6064+16#6061+16#6077+16#606C *);

 CN(44)  ID=+  TYPE=SGDV-E1 DISABLED
       (* RXPDO=16#1600:16#6040+16#60FF+16#6060
          RXPDO=16#1601:16#6040+16#60FF+16#6060
          RXPDO=16#1602:16#6040+16#60FF+16#6060
          RXPDO=16#1603:16#6040+16#60FF+16#6060 *)
       (* TXPDO=16#1A00:16#6041+16#6064+16#6061+16#6077+16#606C
          TXPDO=16#1A01:16#6041+16#6064+16#6061+16#6077+16#606C
          TXPDO=16#1A02:16#6041+16#6064+16#6061+16#6077+16#606C
          TXPDO=16#1A03:16#6041+16#6064+16#6061+16#6077+16#606C *);

VAR
 (* There are no virtual-physical links*)
END_VAR
```

Configuration of the virtual-physical EtherCAT links

The formalism used is in accordance with the standard IEC1131-3. All links between logical devices and EtherCAT addresses must be indicated within the block defined by VAR and END_VAR. With EtherCAT address we are referring to the start position of a sequence of bits inside one of the PDOs of a CN. The length of a PDO is given by the addition of the lengths of the objects that the PDO transfers. The first PDO of a CN has offset 0, while the offset of the next ones corresponds to the addition of the length of the preceding PDOs.

The syntax for the description of a virtual-physical link is as follows (here also the link is finished by the ';' character):

device_name AS EtherCAT_address;

Where:

device_name — Complete name of the logic device. It can be written under the "Group.Subgroup.Device" or "Group.Device" form.

EtherCAT_address — Sequence of characters that identifies precisely an address within an EtherCAT node. The sequence is made in this way:

'%' — first character, obligatory.

'I' or 'Q' — 'I' identifies the address as the address of an input (i.e., transmitted by the CN), 'Q' identifies it as of the address of an output (i.e., received by the CN)

'X', 'B', 'W', 'L' — number of bits associated to the data, received or sent:
'X' = 1 bit, per for the digital inputs and the outputs
'B' = 8 bits, for input and output ports
'W' = 16 bits, for analog inputs and outputs

‘L’ = special character to connect logical axes
If it is missing, you must consider X, i.e 1 bit.

MN_index Number, from 1 on, or alphanumeric identifier showing the EtherCAT bus to which the node is connected

‘.’ Separation character between MN_index and CN_index

CN_index Number, from 1 on, or alphanumeric identifier showing the node

‘.’ Separation character between CN_index and offset

offset Offset with respect to the beginning of the first PDO of the node. It ranges from 0 on and the unit of measure depends on the number of the bits associated to the data, therefore in an EtherCAT address like %QB1.1.3 the byte given begins at the bit 24 of the PDO.

Like in the case of the description of the PDOs, also for the offset it is possible to use the formalism IEC1131-3 to set the numbers.

Example:

```
SERVERIP.Limit1     AS %IX1.100.16;
SERVERIP.Limit2     AS %IX1.100.17;
```

Full example in the paragraph **"Example virtual-physical links"** .

As for the servo drive nodes, the axes are considered as input and output devices and the offset of each axis is the index of the drive inside the node. Statusword and controlword can be connected to logic devices of digital input and output with displacement 16 from an axis to the next one. The significance of each bit of the controlword is set in the AXCONTRI instruction . For the statusword, the significance of each bit is described in the AXSTATUS instruction. We remind you that the offset of the first bit is 0 and not 1.

Example:

```
LTi.X.Ax                AS %IL1.LTi_1.0;
LTi.X.STOP              AS %QX1.LTi_1.2;
LTi.X.SVON              AS %QX1.LTi_1.3;
LTi.X.RESET             AS %QX1.LTi_1.7;
LTi.X.ALM               AS %IX1.LTi_1.3;
LTi.X.WARN              AS %IX1.LTi_1.7;

LTi.B.Ax                AS %IL1.LTi_1.1;
LTi.B.STOP              AS %QX1.LTi_1.18;
LTi.B.SVON              AS %QX1.LTi_1.19;
LTi.B.RESET             AS %QX1.LTi_1.23;
LTi.B.ALM               AS %IX1.LTi_1.19;
LTi.B.WARN              AS %IX1.LTi_1.23;
```

Full example in the paragraph **"Example virtual-physical links"** .

Virtual-physical links in the TRS-CAT

The I/O TRS-CAT device is the equivalent EtherCAT device of the TRS-IO onGreenbus. To this device, that shows digital 16 I/O, you can add IO-E (16 digital I/O), AN-E (an analog input and an analog output), AC-E (encoder reading) expansions, that physically are the same used for the remote Greenbus TRS-IO.

In the basic module, TRS-CAT, the available outputs are 16 starting from the address 0. For the inputs, the first 16 bits have a diagnostic significance and after them 16 available inputs follow. The initial address of the available bits in an expansion is the addition of the bit of the preceding expansions and of the basic module.

Map of the inputs

<table>
<tr><th>Element</th><th>Description of the bits set</th><th>Space</th></tr>
<tr><td>base</td><td><table>
<tr><th>Offset</th><th>Length</th><th>Description</th></tr>
<tr><td>0</td><td>8 bits</td><td>State of each expansion; it can be connected to a port of digital inputs</td></tr>
<tr><td>8</td><td>1 bits</td><td>State BUS</td></tr>
<tr><td>9</td><td>1 bits</td><td>State VOLTAGE</td></tr>
<tr><td>10</td><td>1 bits</td><td>State CURRENT</td></tr>
<tr><td>11</td><td>1 bits</td><td>State NEWMSG</td></tr>
<tr><td>12</td><td>4 bits</td><td>(reserved)</td></tr>
<tr><td>16</td><td>16 bits</td><td>Max 16 inputs, if the outputs are not used. The corresponding bits used as outputs cannot be used as inputs.</td></tr>
</table></td><td>32 bits</td></tr>
<tr><td>IO-E</td><td>Max 16 inputs, if the outputs are not used. The corresponding bits used as outputs cannot be used as inputs.</td><td>16 bits</td></tr>
<tr><td>AN-E</td><td>An analog input</td><td>16 bits</td></tr>
<tr><td>AC-E</td><td>An input encoder, that can be connected to a logic device of counting axis.<table>
<tr><th>Offset</th><th>Description</th></tr>
<tr><td>Number of the expansion from 1 onwards</td><td>Input encoder</td></tr>
<tr><td>16 + Space in bits of all the preceding inputs</td><td>Phase C, i.e. zero position reference</td></tr>
<tr><td>17 + Space in bits of all the preceding inputs</td><td>Quick input</td></tr>
</table></td><td>32 bits</td></tr>
</table>

Map of the outputs

Element	Description of the bits set	Space
base	Max 16 inputs, if the inputs are not used. The corresponding bits used as inputs cannot be used as outputs.	16 bits
IO-E	Max 16 inputs, if the inputs are not used. The corresponding bits used as inputs cannot be used as outputs.	16 bits
AN-E	An analog output	16 bits
AC-E	Cannot be used	32 bits

Example:

```
SERVERIP.Limit1     AS %IX1.100.16;
SERVERIP.Limit2     AS %IX1.100.17;
SERVERIP.CATIN      AS %IB1.100.3;
```

Full example in the paragraph **"Example virtual-physical links"** .
In the case of AC-E encoder counting modules, the value entered corresponds to the expansion number of the TRS-CAT.

Example of virtual-physical link

```
(* The initial part of the file is that indicated in the previous example*)

(*
---------------------------------------------------------------------------
Virtual-Physical Link
*)

VAR

    (* TRS-CAT *)
    SERVERIP.Limit1     AS %IX1.100.16;
    SERVERIP.Limit2     AS %IX1.100.17;
    SERVERIP.CATIN      AS %IB1.100.3;

    (* SERVO LTi_1 *)
    LTi.X.Ax                AS %IL1.LTi_1.0;
    LTi.X.STOP              AS %QX1.LTi_1.2;
    LTi.X.SVON              AS %QX1.LTi_1.3;
    LTi.X.RESET             AS %QX1.LTi_1.7;
    LTi.X.ALM               AS %IX1.LTi_1.3;
    LTi.X.WARN              AS %IX1.LTi_1.7;

    LTi.B.Ax                AS %IL1.LTi_1.1;
    LTi.B.STOP              AS %QX1.LTi_1.18;
    LTi.B.SVON              AS %QX1.LTi_1.19;
    LTi.B.RESET             AS %QX1.LTi_1.23;
    LTi.B.ALM               AS %IX1.LTi_1.19;
    LTi.B.WARN              AS %IX1.LTi_1.23;

    LTi.Z.Ax                AS %IL1.LTi_1.2;
    LTi.Z.STOP              AS %QX1.LTi_1.34;
    LTi.Z.SVON              AS %QX1.LTi_1.35;
    LTi.Z.RESET             AS %QX1.LTi_1.39;
    LTi.Z.ALM               AS %IX1.LTi_1.35;
    LTi.Z.WARN              AS %IX1.LTi_1.39;

    (* SERVO LTi_3 *)
    LTi.Y.Ax                AS %IL1.LTi_3.0;
    LTi.Y.STOP              AS %QX1.LTi_3.2;
    LTi.Y.SVON              AS %QX1.LTi_3.3;
    LTi.Y.RESET             AS %QX1.LTi_3.7;
    LTi.Y.ALM               AS %IX1.LTi_3.3;
    LTi.Y.WARN              AS %IX1.LTi_3.7;

(*
    (* SERVO LTi_4 *)
    LTi.X.Ax                AS %IL1.LTi_4.0;
    LTi.X.STOP              AS %QX1.LTi_4.2;
    LTi.X.SVON              AS %QX1.LTi_4.3;
    LTi.X.RESET             AS %QX1.LTi_4.7;
    LTi.X.ALM               AS %IX1.LTi_4.3;
    LTi.X.WARN              AS %IX1.LTi_4.7;
*)

END_VAR
```

How to write EPLBUS.def file

CANBUS.DEF file is a text format file that describes the hardware configuration and the connections between logical devices and physical devices on POWERLINK. For every module a EPLBUS.DEF file must be written and saved into the configuration folder of corresponding module. (\MODn\CONFIG). Inside the file the part describing POWERLINK hardware should come before the description of the logical-physical connections. The main elements to define the hardware configuration are as follows:

- **(*…*)** beginning and ending of a comment. Comments can be written on more than one text line. You can enter a comment inside another. This is useful when you want to comment a block of definitions whose rows are commented. For example

```
.....
CN(1)  SERVO; (*NODE 1*)
(*
CN(2) SERVO; (*NODE 2*)
CN(3) IO ;   (*NODE 3*)
*)
```

- **MN (number) of attributes** beginning of description's block of a Managing Node (MN). **Number** represents the index used to the [EplAddress] arrangement. Instead of a number you can use an alphanumeric identifier that will be used later on to identify MN in the description bloc of the logic-physical connections. In this case the attribute **ID** is obligatory. A MN is configured by means of the following attributes:
  MASTER: MN gives the signal of synchronism to the others
  ID=number: reference to the MN board position in the PC bus.
  TIME=number: sampling time in msec (it can be 1,2,4,8)
- **CN (number of attributes)** beginning of description block of a Controlled Node (CN). **Number** represents the index used for the [EplAddress] composition. Instead of a number you can use an alphanumeric identifier that will be used later on to identify CN in the description block of the logic-physical connections. In this case the attribute **ID** is obligatory. A CN is considered a part of the POWERLINK subnetwork of the preceding MN description block. A CN is configured by means of the following attributes:
  SERVO: implements the DS402 specification (servodrives)
  DISABLED:disables CN. This word can be entered in any part of the definition, after CN() at the beginning and before ';' at the end
  IO: implements the DS401 (I/O) specification
  ENCODER: implements the DS406 specification (encoder)
  ID=number: CN number. If this attribute is not defined, CN (number) is used
  MPX=mult+slot: if defined, CN is used in multiplexing. **Mult** represents the sampling time multiplier. Following values can be defined: **0**=CN is queried in the asynchronous phase (not realtime);**1**=CN is queried every cycle;from **2 to 16**= CN is queried in multiplexing. **Slot** represents in which slot of time CN will be queried. The range of possible values is between 1 and the value assigned to **mult**.
  RPDO=number: Process Data Object dimension of CN's reception. Value should be between 1 and 1490
  TPDO=number: Process Data Object dimension of CN's transmission. Value should be between 1 and 1490
- **;** ending the description of a MN or CN descritpion block

Below the description of the main elements to define the logical-physical connections:
The formalism used is in accordance with the standard IEC1131. The described data should be located inside the block defined by VAR END_VAR.

- **(*…*)** beginning and ending of a comment. Comments can be written on more than one text line. You can enter a comment inside another. This is useful when you want to comment a block of definitions whose rows are commented.
- **VAR** beginning of block of connections' description.
- **DeviceName** full name of the logical device. It can be written in the form "Group.Subgroup.Device" or "Group.Device"
- **AS** keyword separating **DeviceName** from **EplAddress**
- **EplAddress** shows the hardware address, how many bit employs and which CN is referred to. Its describing formalism is :
  % is the first compulsory character
  I or **Q** is the second character. **I** shows an input device, **Q** shows an output device
  X or **B** or **W** or **L** is the third character. **X** shows the the next value has to be interpreted as a bit and it has to be used in the definition of digital inputs and outputs. **B** shows that the next value should be interpreted as a byte and it has to be used in the definition of digital input and outputs ports.
  W shows that the next value should be interpreted as a word and it has to be used in the definition of digital input and outputs ports. **D** shows that the next value should to be interpreted as 32 bit and it has to be used in the definition of analog input and outputs ports. **L** shows the the next value should be interpreted as 8 byte and it has to be used in the axes definition. If omitted, the next value is interpreted as a bit. The following characters are a sequence of figures, divided by a point '.' , showing the address. The first number refers to MN, the second one to CN, the third, optional, is an offset inside CN. This offset can also be expressed on 2, 8 or 16 according to the IEC formalism. If the offset is omitted, a value equal to 0 is considered.

- **;** ending the connection's description
- **END_VAR** ending of block of connections description

Whatever is found after the keyword END_VAR of end block is ignored.
Correctness of the file from both points of view of syntax and content is verified at the startup of Albatros. If an error is found, an error message is displayed. All the errors described are in the file ERREPL.TXT saved in the folder provided in Tpa.ini under DirReport.

Example:

```
MN  (1) ID=142332 TIME=1 MASTER;

        CN (1)                      SERVO       MPX=1       RPDO=4      TPDO=8;
        CN (2)                      SERVO       MPX=1       RPDO=4      TPDO=8;
        CN (3)      ID=17           IO          MPX=2       RPDO=4      TPDO=8;
        CN (4)      ID=21           IO          MPX=4+1     RPDO=4      TPDO=8;
        CN (5)      ID=22           IO          MPX=4+2     RPDO=4      TPDO=8;
        CN (6)      ID=108          ENCODER     MPX=0       RPDO=4      TPDO=8;

VAR
      Main.EV1                AS %QX1.3.10;
      Main.EV2                AS %Q1.3.11;
      Main.Axes.InpPort       AS %IB1.5.12;
      Emerg.InputW            AS %IX1.5.13;
      Axes.AxisX              AS %IL1.1;
      Axes.AxisY              AS %IL1.2;

END_VAR
```

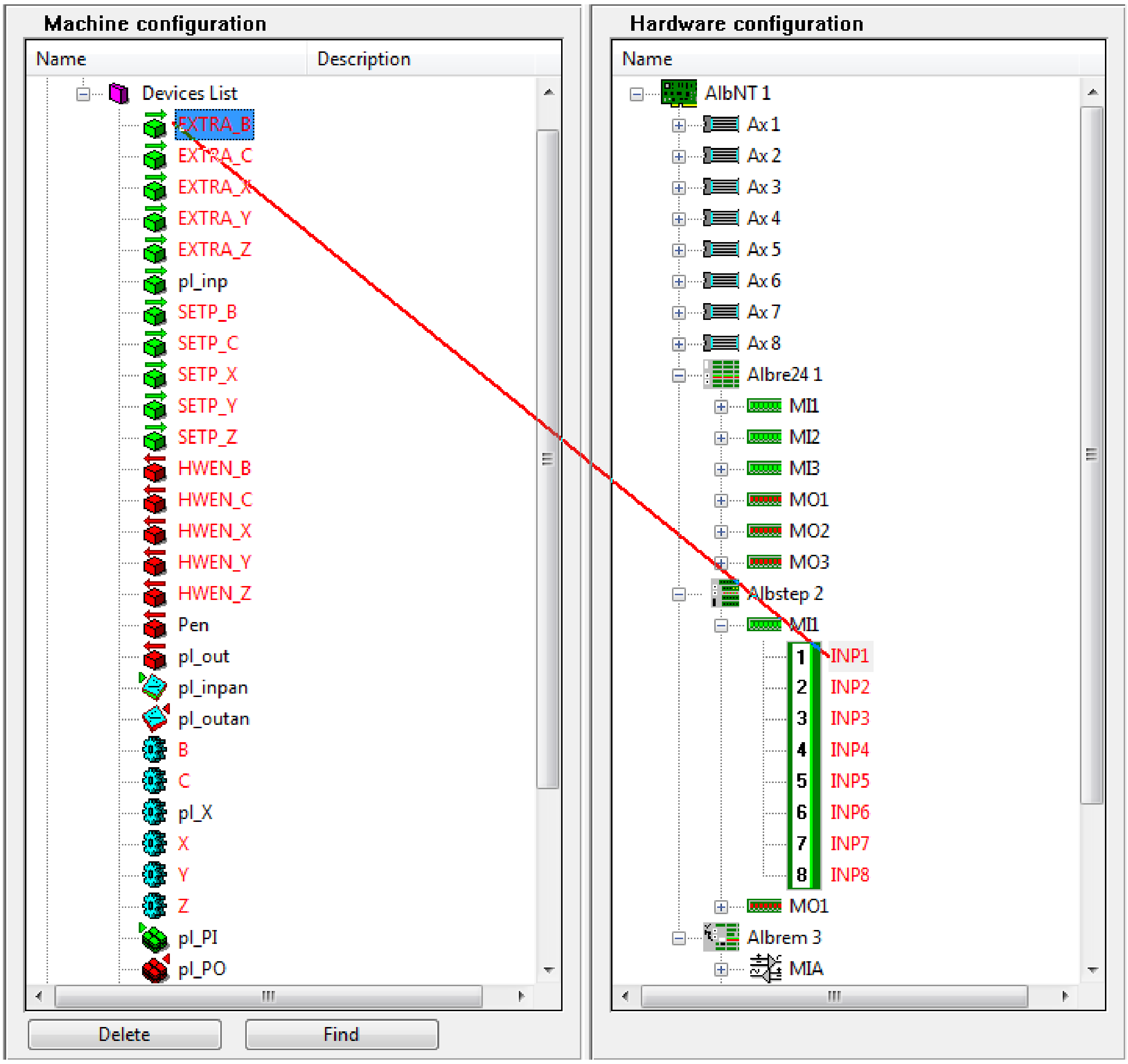
1.4.3 Virtual physical Configuration

Virtual physical Configuration is the last configuration step and consists in connecting the logic devices to the hardware components.

For each axes of a Mechatrolink II board 6 inputs and 1 digital output can be configured in virtual-physical. For a detailed description, please, read chapter **GPL Language->Instruction->Mechatrolink II->MECGETSTATUS.**

If Ether-CAT bus is available in a module, you can anyway configure some boards for the Mechatrolink II bus, but with some restrictions: with 1 ms realtime you cannot connect more than 6 Mechatrolink II axes (each bus); with 2 ms realtime, the restriction rises to 16 axes.

Opening the Virtual physical Configuration two windows are displayed: the Machine Configuration window (virtual) on the left, and the Hardware Configuration window (physical) on the right. Both show a graphic representation of all the elements composing the system in a tree structure.



Virtual-Physical Configuration

The existing virtual-physical connections are highlighted in the "Machine Configuration", by the Name of the device (in red), while in the "Hardware Configuration" window they are highlighted by the name of the type of signal, which follows the number of the terminal, also in red.
If in the system some devices are configured on CAN, POWERLINK and EtherCAT buses, they are displayed in fuchsia and they cannot be modified. All this because bringing together the logical device and the physical device must be defined in the external .DEF files.
The devices or the terminals still to be connected are marked in black.
The signals indicating the axes, in the "Hardware Configuration" window, are all preceded by a rectangle whose colour corresponds to the colour of the sheathing of the wire inside the connection cable.
It is possible to highlight a connection by selecting a logic device (or a hardware component) and pressing the space bar: the connection is shown as a red line between the device and the hardware component. It is also possible to keep the connection visible at all times by pressing **[Alt+Enter]**.
To show which logic device is connected to the hardware component, select the hardware component and double click on it with the mouse.
To select the logical device and the physical device to connect various procedures are possible:

First procedure

- Display on the screen, through the "Hardware Configuration" window, the physical terminal to which the device has to be connected.
- Select, or point, the logical device required in the "Machine Configuration" window.

Second procedure

- Select, or point, the chosen virtual device in the "Machine Configuration" window.
- Select the command from **Edit->Find the suitable physical device** menu or **[CTRL+space]** key combination . Albatros displays automatically in the "Hardware Configuration" window the first

physical unengaged device to which the logical device can be connected.

Third possible procedure

- Select, or point, a virtual device in the "Machine Configuration" window.
- Select the command from the menu **Edit->Find next unlinked device** or the shortcut key **[CTRL+NumPad+]** or the command **Edit->Find previous unlinked device** or **[CTRL +NumPad-]** keyboard shortcut.

To connect the two selected devices:

- Click on the logical device to connect with the left hand button of the mouse, and keeping it pressed, drag it towards the selected terminal. A red line will appear to indicate connection in progress. When you have reached the terminal line, release the button to terminate the operation or
- select the command **Link!** from the menu **Edit** or the keyboard shortcuts **[CTRL+L]**.

To remove a connection, select the device or the affected component and press the button **[Remove]** or the button **[Delete]** on the keyboard.

1.4.4 Cabling maps

When the virtual devices and the corresponding physical devices have been connected, it is possible to print maps or lists of the virtual-physical connections.

To perform this operation it is necessary to have installed MS-Word (version 6 or later) on the system, as Albatros uses its functions to format the maps.
The system must also have been configured correctly, which means that the system must have the model files used for map compiling. These are a series of files with a ".doc" extension which normally lie in the System folder or in another installing folder (often the "Map" file). The important is that the folder where these files lie corresponds to the one specified in the ***TPA.INI*** file, key: "DirMaps". For example:

```
[TPA]
DirMaps=C:\Albatros\Maps
```

To print the cabling maps, select any hardware component in the right hand window of the Virtual-Physical configuration or in the window of the Hardware configuration.

Press the Print icon in the Status Bar, or select the heading ***Print*** from the ***File*** menu; the usual print options window will appear. When the printer is set to your satisfaction, confirm by pressing **[OK]** and another window will show the list of hardware components present in configuration.
Select from this window all the components to be included in the cabling map. To select more than one component, select the components with the mouse while keeping the "**Ctrl**" key pressed.
Click on **[OK]** and the cabling maps will be printed. If the **Print on paper** option is deselected, the maps will be saved as MS- Word documents in the file of the current module (Mod.0, etc).

Because of the large number of pages which are often necessary for printing, we suggest printing a proof sheet, with only one hardware component, to check that everything is working. If a list of logic devices is printed instead of the map, probably no component (for example an axis card or remote) was selected in the hardware window. When a component is selected, its name appears highlighted in blue.

1.4.5 List of navigation keys to navigate through a tree structure

Key	**Description**
Up Arrow Down Arrow	moves the selection to the immediately previous row or to the following one
Right arrow	expands the selected branch to an extra level and, if already expanded, moves the selection on the next branch
Left arrow	collapses the selected branch and, if already collapsed, transfers the selection on the previous branch
+	expands the selected branch to one level
-	collapses the selected branch
*	expands all the levels of the selected branch

2 Development tools

2.1 Editor GPL

2.1.1 GPL Editor functions

GPL editor is the instrument that allows you to create and modify the files in the Albatros GPL code. This function can only be activated as from the manufacturer password level. Each functions file contains information which can be displayed in the ***File->Information*** menu.
The functions are the ones typically used in a text editor, so we find commands such as ***Copy***, ***Paste***, ***Find***, ***Replace*** etc. All these commands can be selected from the menu ***Edit.***

Undo if possible, erases the last operation performed. The situation is reverted to the older state, before the last operation performed.

Redo The situation is reverted to the older state preceding the last Undo command.

Cut Text or selected data are removed and copied in a temporary memory to enable their possible insertion with the command *Paste*

Copy Text or selected item is copied in a temporary memory to be inserted again with the command. *Paste.*

Paste Temporary memory content is inserted using different criteria according to the active function.

Delete Text or rows or the selected item are deleted. Deleted data can be recovered by acting immediately upon the command *Delete*

Select All allows the whole text of the active file to be selected. To the selected rows Copy, Cut, Paste commands can be applied.

Find... searches a text in the current document. You can set some criteria to use under research such as search direction and case-sensitive distinction.

Find next permits the repetition of a previous search, enabling the change of the research criteria, set by with the command Find.

Replace allows you to search a text of the current document and to replace it with another text.

Insert device inserts a device by selecting it from the list of the devices. This function is particularly useful when you work with a large number of devices whose name can be difficult to remember. Only the devices of the current module that can be recalled and all the public devices of the other modules are displayed.

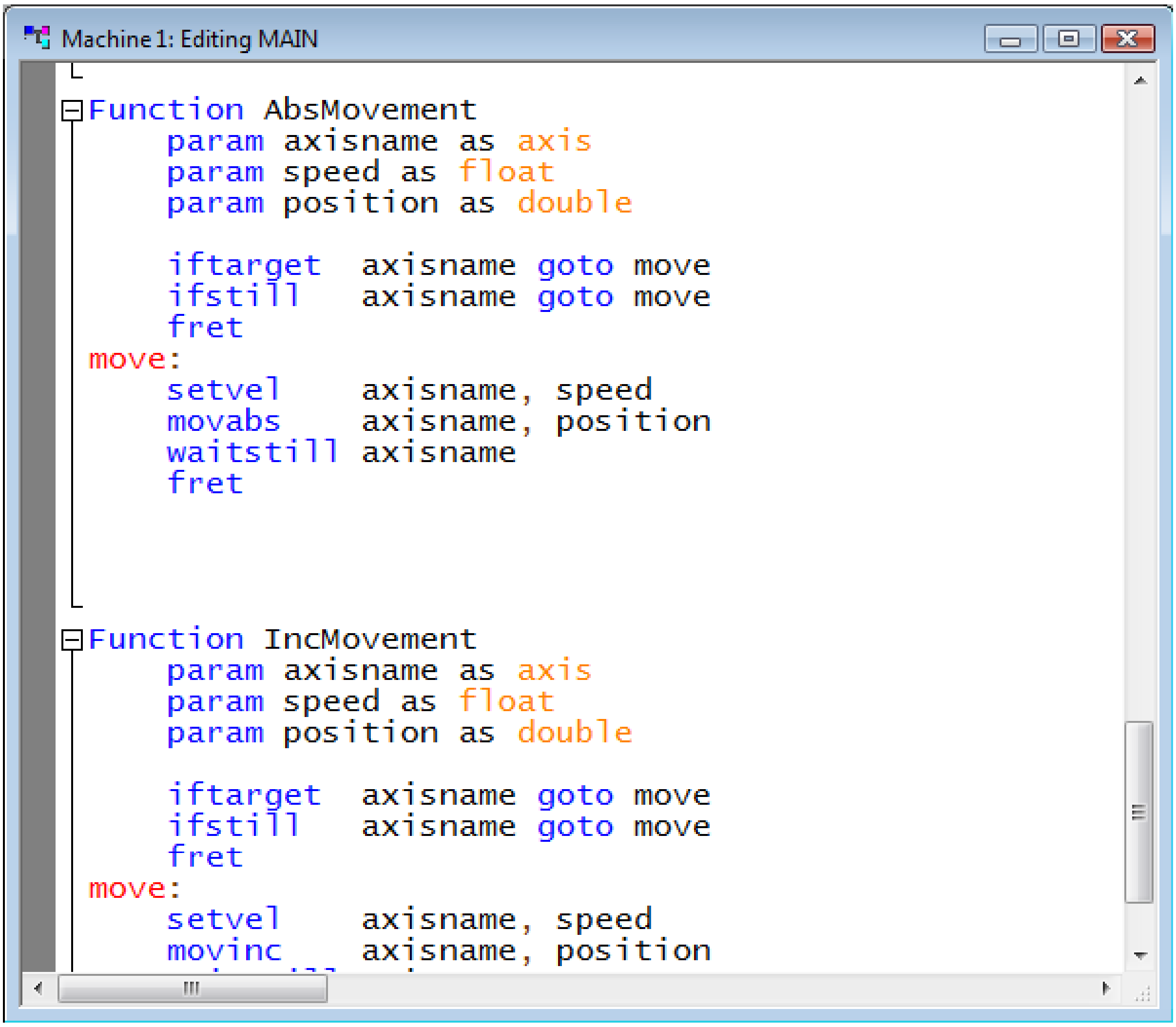
Insert function inserts an empty function including some comments to use as a guide in Edit.
It inserts a function or a part of a function starting from the position of the cursor. The function is read by a prototype file, written from the machine constructor. More prototype files can be written. A prototype file is a text file, whose name must start with the GPL prefix and TXT extension. It must be stored in the directory, where the libraries are normally stored (usually system \lib). If more prototype files are defined, selecting a command, a dialog box is opened, in which the list of the prototype names is displayed without prefix and without extension.
Prototype files can contain, for instance, const definitions commonly used, handling functions of system errors, generic functions, codes implementing algorithms for various usages, and so on. They also content some comments.
A prototype file can be created by saving the selected text in the file of GPL functions. This command is available only as keyboard accelerator **[**Ctrl+Shift +C**]**. A dialog box opens to insert the name that has be given to the code fragment.

Insert message... inserts in the GPL text the numeric code associated to the chosen message. Enables some new messages to be entered in the language files.

Enable/Disable new page inserts or removes a page break ▤. Page break can be used as a bookmark to spring to remarkable positions inside the function file.

Enable page break after moves edit cursor to the row of the next page break with respect to its position

Enable page break before moves edit cursor to the row of the previous page break with respect to its position



```
Function AbsMovement
    param axisname as axis
    param speed as float
    param position as double

    iftarget  axisname goto move
    ifstill   axisname goto move
    fret
move:
    setvel    axisname, speed
    movabs    axisname, position
    waitstill axisname
    fret



Function IncMovement
    param axisname as axis
    param speed as float
    param position as double

    iftarget  axisname goto move
    ifstill   axisname goto move
    fret
move:
    setvel    axisname, speed
    movinc    axisname, position
```

GPL Editor

Syntax corrections are carried out in the archiving phase, when the text is also compiled. However, the programmer can easily make a preliminary inspection, as the text is displayed in different colours according to what it represents. For example, instructions are in blue, comments in green and labels in red.
The value of tabulations for the initial position of the GPL code, the initial position of the first subject of instructions and the initial position of the comment, can be modified using the ***Options->Tabulations...*** menu.
Tab value can be modified from menu ***Options->Tabulations...*** Two types of tabulations can be defined:

- absolute tabulations: they set the initial position for the instructions of GPL code the initial position of the first argument of the instructions and the initial position for the comment.
- relative tab (spaces): it sets how many spaces is a tab

Tabulations also help to make the lay out of the GPL code more immediately comprehensible.

Each instruction or keyword is linked to the online help for further support when editing a function. To recall the help simply place the cursor on the instruction and press **[F1].**

Each line of text can contain only one instruction. To continue the instruction in the following row press the character '_' (preceded by a space) as the last one of the row. This allows you to insert comments in the middle of an instruction:

```
Message                                                  _
        1000    ;code of the message that will be displayed     _
        3       ;synoptic cell in which it will be displayed    [Enter]
```

2.1.2 Avalaible keyboard shortcut list

Clearing a text

Key	Description
Backspace	erases a character on the left or the selected text
Ctrl+Backspace	erases the word on the left
Del	erases a character on the right or the selected text
Ctrl+T	erases the words or the spaces on the right
Ctrl+Del	erases the word on the right and all the following spaces until the beginning of a new word

Comment of more text rows

Key	Description
Ctrl+';'. In the Italian keyboards [Shift] key must be pressed as well	this adds or removes the comment characters to the selected rows.

Cursor positioning

Key	Description
Up arrow	moves the cursor to the selected direction
Down arrow	
Right arrow	
Left arrow	
Home	moves the cursor to the beginning of the row to the beginning of the row and to the first character of the row alternately
End:	moves the cursor to the end of the row
Ctrl+Home	moves the cursor to the beginning of the document
Ctrl+End	moves the cursor to the end of the document
Ctrl+Left Arrow	moves the cursor by one word on the left
Ctrl+Right Arrow	moves the cursor by one word on the right
Ctrl+Enter	moves the cursor on the first character of the following row

Select

Key	Description
Shift+Home	selects from the cursor position until the beginning of the row
Ctrl+Shift+Home	selects from the cursor position until the beginning of the document
Ctrl+Shift+End	selects from the cursor position until the end of the document
Ctrl+Shift+Left Arrow	selects the word or the the spaces on the left of the cursor
Ctrl+Shift+Right Arrow	selects the word or the the spaces on the right of the cursor
Shift+Page Up	selects a page up from the current position of the cursor
Shift+Page Down	selects a page down from the current position of the cursor
Ctrl+W	selects the word where the cursor is placed
Ctrl+A	selects the whole document

Rectangular selection

Key	Description
Alt+	selects a rectangular code group
Shift+Up Arrow	
Shift+Down Arrow	
Shift+Left Arrow	
Shift+Right Arrow	

Tabulations

Key	Description
Tab	in case of unavailable selected text, it inserts spaces between characters, as defined in ***Options->Tabulations.*** If many rows have been selected, *Tab* inserts on the right the spacing set for the relative tabulation.
Shift+Tab	In case of unavailable selected text, *Shift+Tab* moves the cursor on the left side or the spacing defined ***Options->Tabulations.*** If one or more rows have been selected, they are moved to the left side of the spacing set for the relative tabulation.

⊟ **Copy and Paste**

Key	Description
Ctrl+C	copy the selected text into the Clipboard
Ctrl+Ins	
Ctrl+X	deletes the selected text and copy it into the Clipboard
Shift+Del	
Ctrl+V	inserts the content of Clipboard from the cursor position
Shift+Ins	
Ctrl+Y	eliminates the row where the cursor is placed and copies its content into the Clipboard
Drag'n'drop (with the mouse)	the selected text is draged and moved to the new position after its release
Ctrl+Drag'n'drop (with the mouse)	the selected text is draged and copied to the new position after its release

⊟ **Cancel / Restore**

Key	Description
Ctrl+Z	cancels the last typing
Alt+BackSpace	
Ctrl+Shift+Z	restores the last typing

⊟ **Search and Replace**

Key	Description
Ctrl+F3	searches down into the whole document for the word which the cursor is placed on.
Ctrl+Shift+F3	searches up into the whole document for the word, which the cursor is placed on.
F3	searches for the following occurrence. The dialog box **Find** should be closed.
Shift+F3	searches for the previous occurrence. The dialog box **Find** should be closed.
Alt+F3	opens the dialog box **Find** and as a text to be searched sets the word, which the cursor is placed on.

⊟ **Displaying compilation errors**

Key	Description
Double-click on the error	places the cursor on the row of the GPL function where the error described occurred
F4	places the cursor on the row of the GPL function where occurred the error, that follows the last selected error.
Shift+F4	places the cursor on the row of the GPL function where occurred the error, that precedes the last selected error..

⊟ **Creating a prototype file**

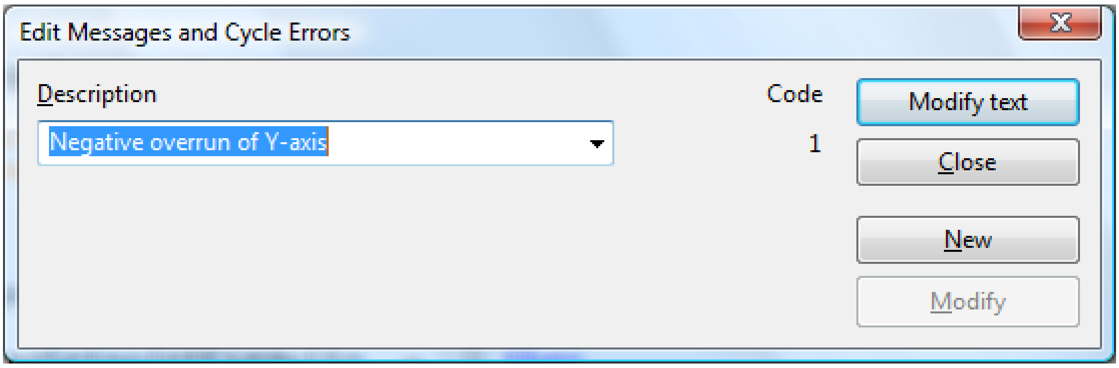
Key	Description
Ctrl+Shift+C	saves the text selected in the file of GPL functions. A dialog box opens to insert the name that has to be given to the code.

Folding control

Key	Description
Ctrl+M	expands or collapses the selected folding.

2.1.3 Insert Message

Albatros uses two kinds of messages: module messages and group messages. The command can be selected from the menu ***Edit->Insert Message.***
Group messages are inserted directly in editor when writing the GPL code, by using the DEFMSG instruction. These messages can be displayed and used only inside the group in which they are defined, so that the same message definition can be used in various groups, without creating superimposition.
Module messages, unlike group messages, can be used by any group. They can be inserted through the dialog window that allows both recalling any existing message from the language file and introducing new messages.


Edit Messages and Cycle Errors

Description: Negative overrun of Y-axis

Code: 1

Message management window

Using this procedure avoids having to pass to Winmess.exe and worrying about opening the right file. The message will be inserted in the current language although, later, it will have to be translated in the other languages (this time using Winmess.exe).
All the messages in the language file are listed under the heading **Description.** To insert a message in the function, choose the required text and select the **[Modify text]** button.
To modify an existing message **[Modify]** or create a new one **[New]**, first type in the modification or the new text and then press the corresponding button.

2.1.4 Cryptography

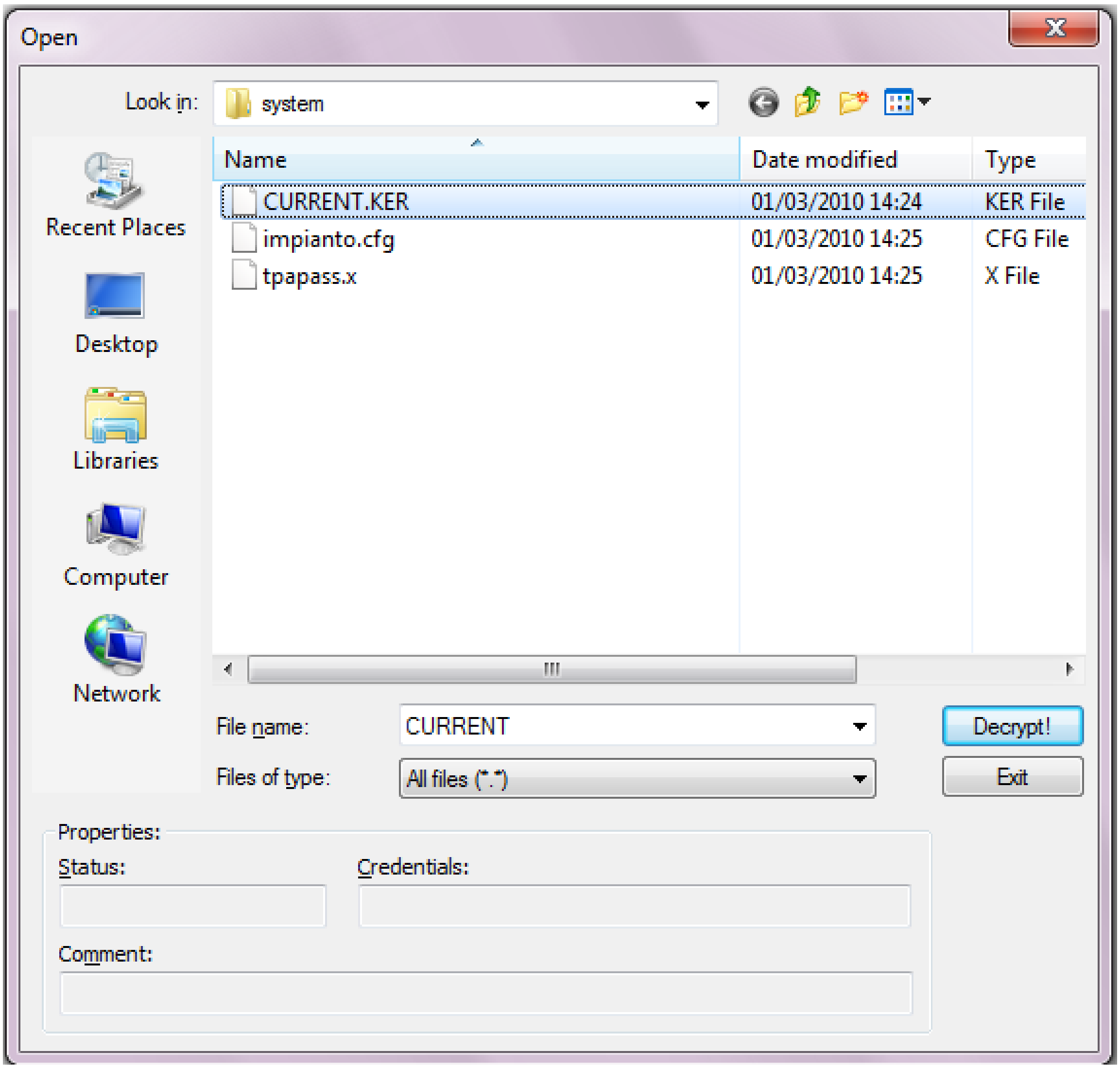
In Albatros it is possible to use encryption so that the source text of functions cannot be displayed.

Cryptography is enabled by selecting Tele+=0 or 1 in TPA.INI. The default value is 0. In this case, when Albatros saves a functions file, the save mode does not change.

When a functions file is saved and cryptography is enabled, the following message will be displayed: "Do you want to encrypt the file?". If you choose no, the file will be saved as plaintext. A previously saved, plaintext file can subsequently be encrypted, while an encrypted file will not change, and will be saved in the same way by default.

When a functions file is saved for the first time, with cryptography enabled, and a daily Manufacturer password is used, the file will not be encrypted, but only saved as plaintext.
Subsequently, the encrypted functions file may only be displayed or edited in Albatros by the user who previously saved it. The owner of an encrypted functions file cannot change!

The external file SBIANCA.EXE must be used to decipher the file. This is located in the Bin folder of Albatros. When the programme is run, the following window is displayed:



Look in: system

Name	Date modified	Type
CURRENT.KER	01/03/2010 14:24	KER File
impianto.cfg	01/03/2010 14:25	CFG File
tpapass.x	01/03/2010 14:25	X File

File name: CURRENT

Files of type: All files (*.*)

Status:

Credentials:

Comment:

In this window, files to decrypt can be selected. The Status and Credentials are displayed for each file. The status may be "Plaintext" or "Encrypted".
"Credentials" gives information about file visibility. "Freely readable" means the file can be displayed from the current password level. Blocked means the file cannot be displayed.
Select the files, then click on "Decrypt!" to decipher them.

2.2 Libraries

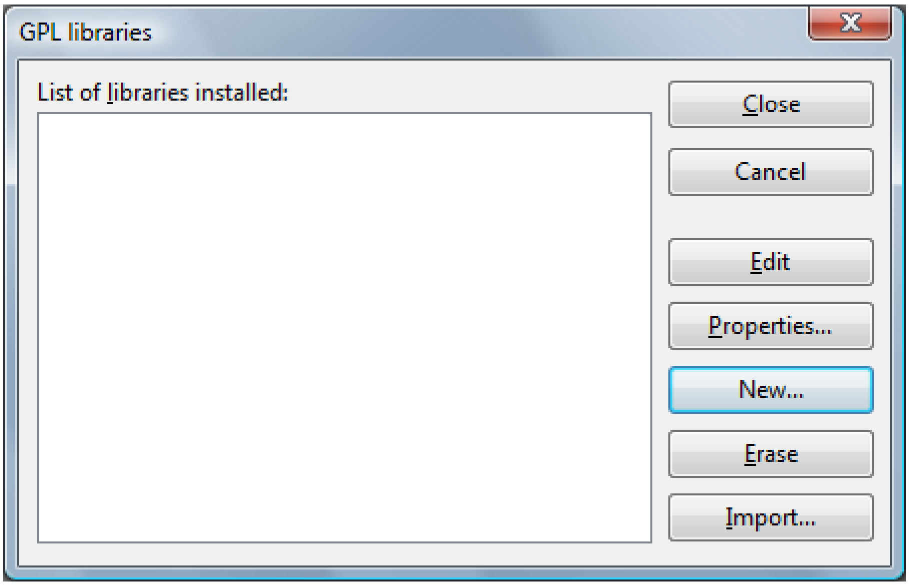
2.2.1 Create and modify

A library is a collection of GPL functions which can be called within the custom GPL code without being limited to a particular configuration. Libraries are very useful, as they can be easily copied from one machine to another, which avoids having to rewrite common code when implementing new machines. For example, we could create a mathematical and geometrical functions library.

Library files are archived in the system\lib folder. They are compiled by executing one of the following commands: ***CNC->Initializing***, ***File->Compile All***, ***Save*** library file or global variables file.

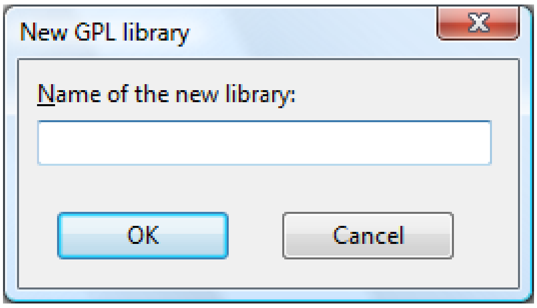
If in the GPL code a machine is given a function or variable name which already exists in a library, in the compiling phase the machine will always have the priority. If the same name is used in two different libraries, when writing the GPL code, we suggest using the following full syntax to identify the required one: **namelibrary.namefunction**. For example, if the LengthSegment function appears both in the LIBGEO library and the LIBMAT library, and we want to identify the function belonging to the LIBGEO library, we write:

LIBGEO.LengthSegment.



GPL library management window

All the operations concerning the library are managed through the dialog window above. It is possible to create new libraries **[New]**. The name given to the library will be added to the list of libraries installed. Moreover it is also possible to import already existing libraries and to transform files of groups into a new library; this is done by recalling them through the dialog window opened by pressing **[Import..].** The same operation is used to recover libraries which had previously been eliminated with the **[Erase]** command.

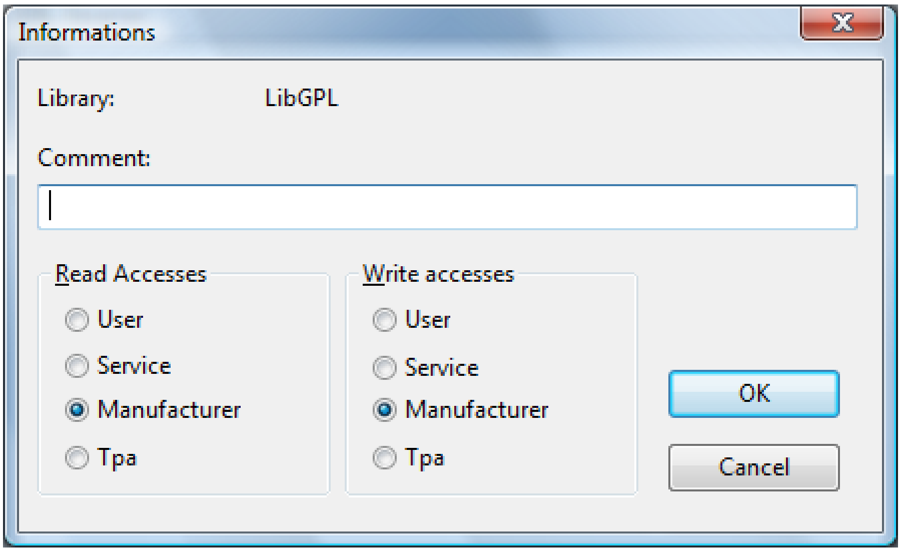


New library

To modify the code of a library, select the **[Edit]** button. The library is opened by GPL editor. When writing the library functions remember these basic rules:

- it is not possible to access devices, functions, and variables belonging to the configuration in which the function is being written.
- it is possible to call public functions and variables from other libraries.
- the functions declared inside a library are defined as private by default. To make it possible for other function files to recall them, they have to be declared as PUBLIC.

Library modification is subject to access level limitations of the person using Albatros. It is possible to assign or modify library access authorisations by selecting the **[Properties]** button.



Informations

Library: LibGPL

Comment:

Read Accesses
○ User
○ Service
● Manufacturer
○ Tpa

Write accesses
○ User
○ Service
● Manufacturer
○ Tpa

Library properties

Any global variables declared in a library are displayed in a section of Diagnostic. The display of library elements depends on the access rights of the person using Albatros.

2.3 Debug

2.3.1 The debugger

The debugger is a function of Albatros which allows you to follow the sequence of instructions of a GPL task step by step, thus allowing you to identify and correct any logic errors and anomalous behaviour of the code.

This function can only be activated from the manufacturer level or a higher password level.
The debugger allows the user, for example:

- to assign breakpoints
- to interrupt the execution of a task and display the value of a variable
- to supervise the execution sequence of a function
- to check the value adopted by a local variable
- to check that, in the case of an instruction, the right branch was chosen

The commands required in debug mode can be selected from the ***Debug*** menu. The main ones are:

Go resumes the execution of a blocked task. The task will continue until the end, it will not be stopped again or an interruption point will not be.

Restart restarts the debug of the current task

Break now stops the execution of the task which is being debugged. The cursor is placed at the row, where the instruction has been broken.
Once the task has been stopped, its execution can be piloted and the status of the local variables can be checked.

Step into steps into a single GPL instruction The task should have been previously broken.

Step out carries out all the instructions until the first instruction after the current one

Step over carries a single GPL instruction out or, if the instruction is a function call, it carries the whole instruction out

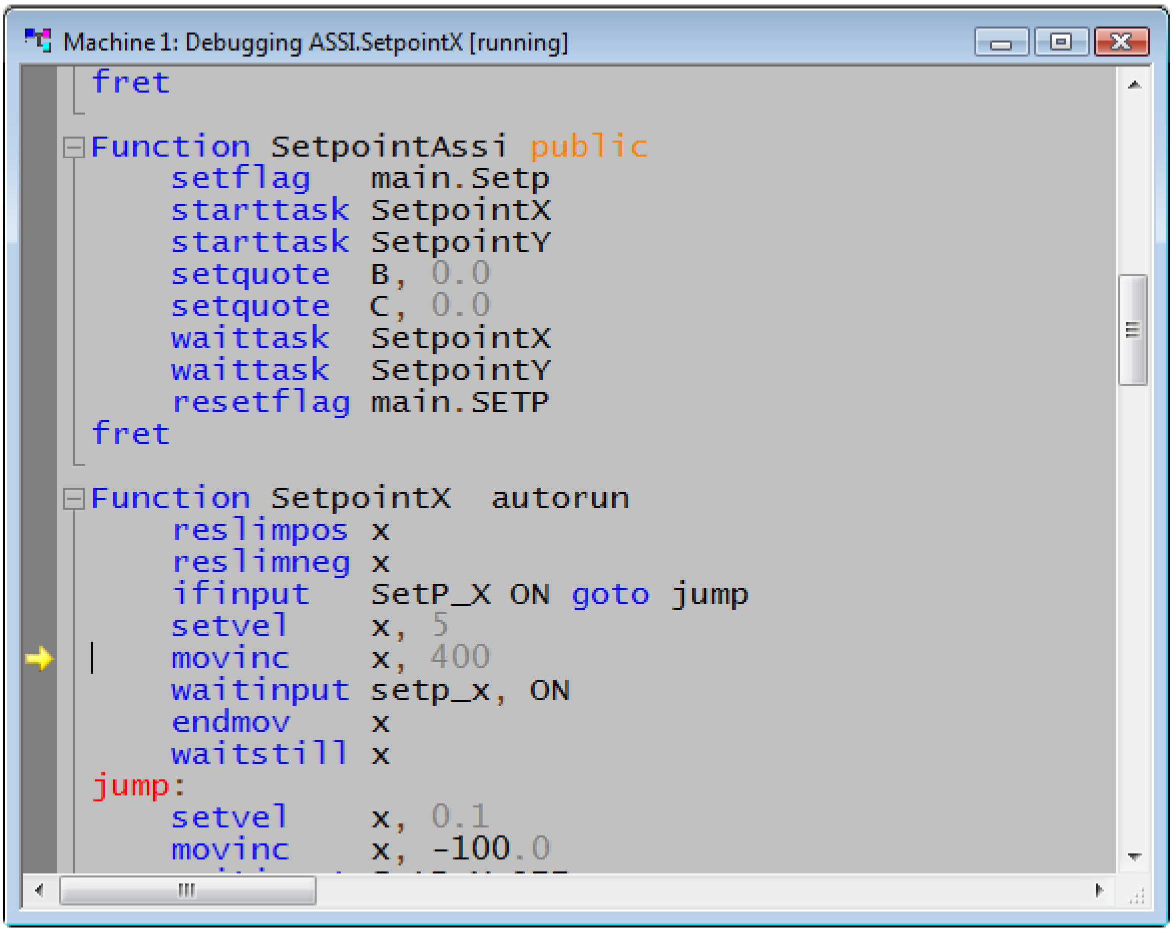
Step to Cursor carries out the instructions until the cursor position

End debug usage. The function file that was being debugged is opened in Edit mode.

To access the debugger, display the list of tasks in execution (from the menu ***Debug->Task in execution*** or the list of All tasks (from the menu ***Debug->All tasks***) and then select the task to be debugged.

Before executing the debug make sure there are no function compiling errors (for example: syntax errors and undeclared variables) and that the module to be debugged has been started correctly.

The debug window is similar to the GPL editor window, however it does not allow you to modify the code. The background of the window is grey and the line in execution is highlighted in yellow.



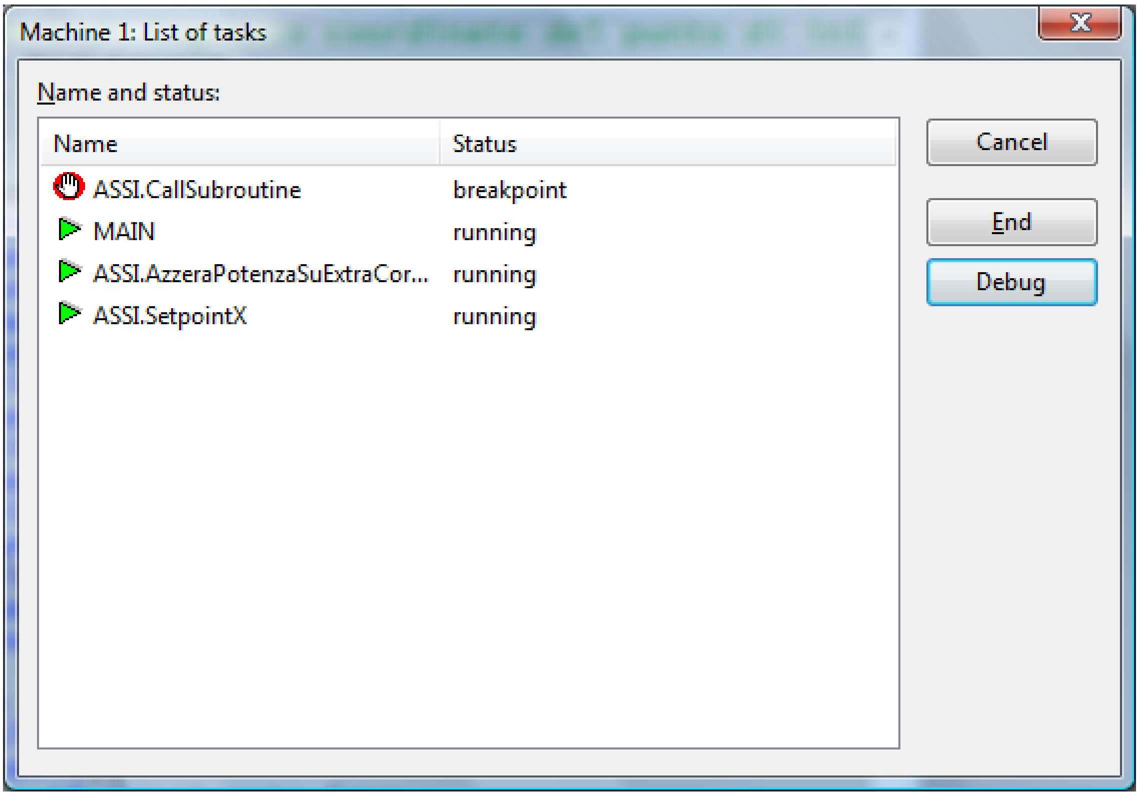
```
fret

Function SetpointAssi public
    setflag   main.Setp
    starttask SetpointX
    starttask SetpointY
    setquote  B, 0.0
    setquote  C, 0.0
    waittask  SetpointX
    waittask  SetpointY
    resetflag main.SETP
fret

Function SetpointX  autorun
    reslimpos x
    reslimneg x
    ifinput   SetP_X ON goto jump
    setvel    x, 5
    movinc    x, 400
    waitinput setp_x, ON
    endmov    x
    waitstill x
jump:
    setvel    x, 0.1
    movinc    x, -100.0
```

Debugger window

Notice: It is not possible to debug simultaneously more than one task belonging to the same module.

2.3.2 Task in execution

The command can be selected from the menu ***Debug->Task in execution.*** It displays the list of tasks in execution associated to a machine or module. It is possible to execute the debug or interrupt execution of a task by selecting the task and clicking on the **[Debug]** or **[End]** button, accordingly.

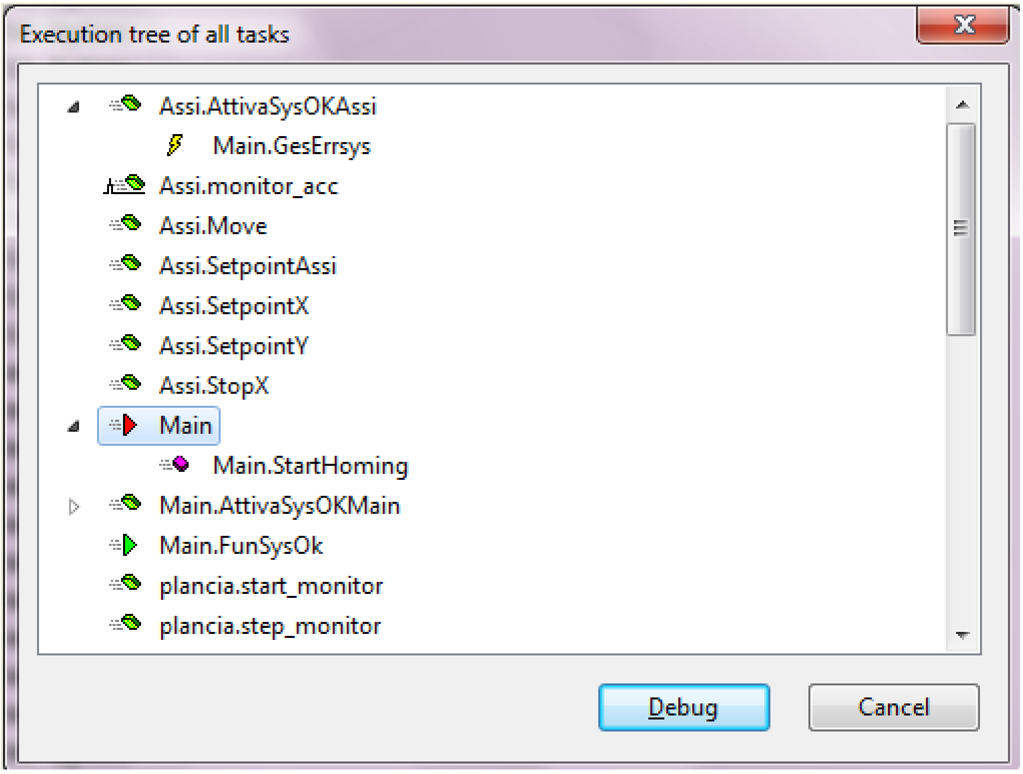


Name	Status
ASSI.CallSubroutine	breakpoint
MAIN	running
ASSI.AzzeraPotenzaSuExtraCor...	running
ASSI.SetpointX	running

List of active tasks

2.3.3 All tasks

It displays in a dialog window the list of all the tasks defined in the GPL code. These are represented graphically as a tree structure, as shown in the figure below. When we select a function, the file in which it is defined is opened and the curser is positioned on the first instruction of the function. This allows you to set Breakpoints even before starting execution.
It is important to select the function from the task branch we want it to be called from.



Execution tree of all tasks

- Assi.AttivaSysOKAssi
  - Main.GesErrsys
- Assi.monitor_acc
- Assi.Move
- Assi.SetpointAssi
- Assi.SetpointX
- Assi.SetpointY
- Assi.StopX
- Main
  - Main.StartHoming
- Main.AttivaSysOKMain
- Main.FunSysOk
- plancia.start_monitor
- plancia.step_monitor

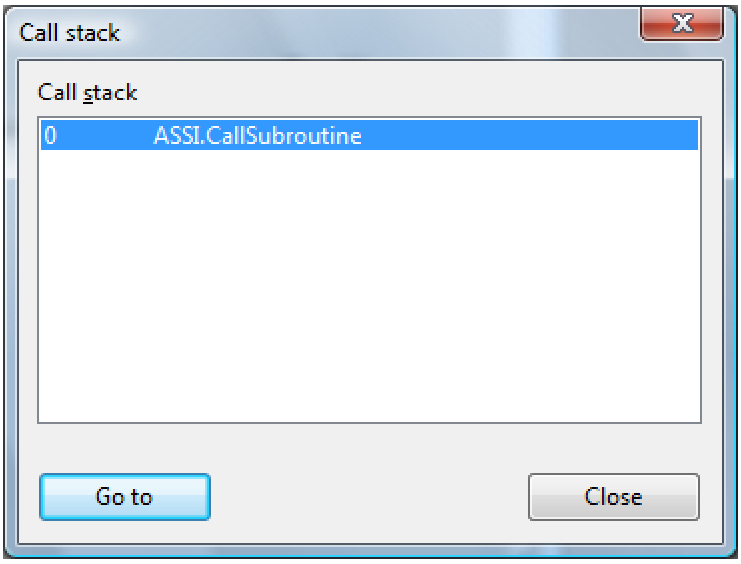
List of tasks

Below we describe the meaning of the symbols used in the composition of the task execution tree. An interesting symbol is the one indicating the recursive function, that indicates a function which includes a recall to the function from which it is called.

Symbol	**Description**
▶	task of the Intergroup's main function
▶	autorun task
●	generic task
⏱●	real-time task
●	group function
⚡	group function executed by instructions such as ONINPUT, ONFLAG
🗀	library function
⚡🗀	library function executed by instructions such as ONINPUT, ONFLAG.
↩	recursive function

2.3.4 Show call stack

During debug it is possible to display the list of functions which have been called but still haven't returned (that is, all the functions in which the FRET instruction has not yet been executed). A dialog window appears, listing all the function calls leading to the current instruction. The function executed last is at the top of the list.



0 ASSI.CallSubroutine

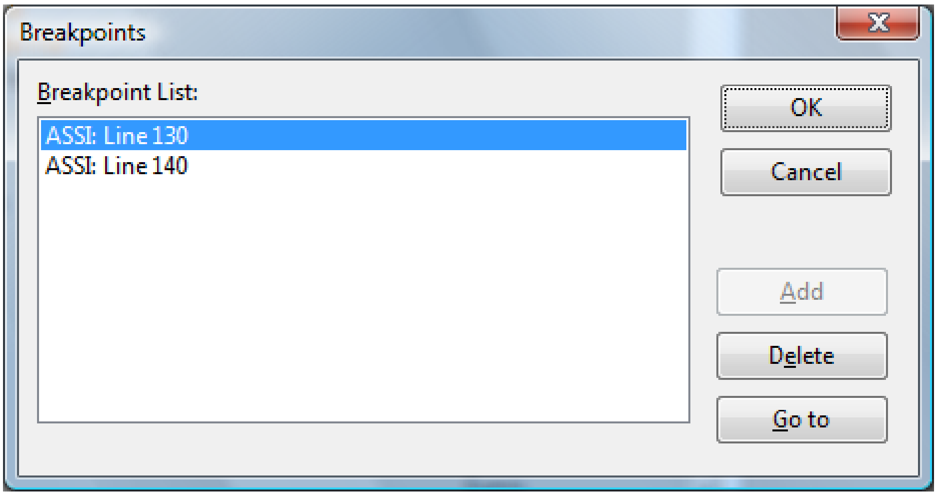
List of all function calls

To observe the behaviour of a function call:

- move the curser to the desired position in the function
- select ***Debug->Step to cursor*** to take program execution to the desired position
- select ***Debug->Show Call stack***, or the shortcut button **[CTRL+K]**.
- the name of a function can be selected from the Call stack dialog window. The cursor will then go to the first instruction of the chosen function.

2.3.5 Breakpoints

A breakpoint allows you to examine all the details of an instruction execution sequence, to examine or modify variables and devices, to examine the list of function calls etc.
Task execution is interrupted when the instruction containing the breakpoint is reached.
Breakpoints can be set both before executing a certain task and during execution (from the menu ***Debug->Breakpoints***). It is also possible to delete the breakpoints when they are no longer necessary.



ASSI: Line 130
ASSI: Line 140

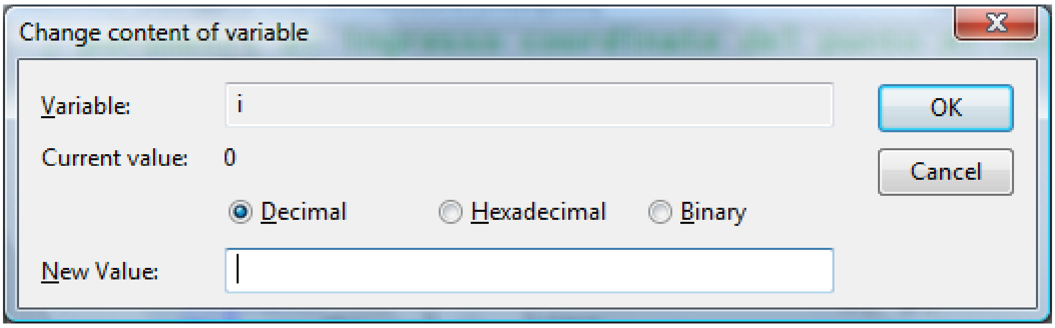
List of breakpoints

In certain situations, despite having inserted breakpoints the task is not interrupted, because execution never reaches the breakpoint. In this case the task can be interrupted by using the command: ***Debug->Break now***. The cursor will be positioned on the GPL instruction which was about to be executed when the task was interrupted.

2.3.6 Variable content

This command can be selected from the menu ***Debug->Content of variabile.***
After interrupting task execution the following can be displayed:

- the value of the local variables declared in the function where the task has been interrupted
- global variables
- the value assumed by an expression
- the state of devices and device parameters



Change content of variable

Variable: i

Current value: 0

● Decimal ○ Hexadecimal ○ Binary

New Value:

Display/Change content of a variable

If the variable (or device) in not read-only, its content can be modified: obviously any modifications will affect the execution of the next task.
Changing the value of a variable or device allows you to test execution in different conditions from usual, to correct errors and carry on with the execution of the next instructions.

It is possible to display the content of a variable, of a device or of a constant also by moving the mouse on the variable, on the name of the device or on the constant. A tooltip is displayed, where the type, the name and the value of the data is shown. If you select an expression, its result is displayed. If the mouse pointer is inside the selection, the whole selection is used, otherwise only the word where the mouse pointer is placed. If the mouse pointer is not inside a word, the whole argument is used.
E.g., to see the value of the Mx[3][column], if the mouse pointer is on "3", 3 is displayed in the tooltip; if the mouse pointer is on "column", the value of the column is displayed; if it is on "matrix" nothing is displayed; if it is on a square bracket, the value of Mx Mx[3][column] is displayed.

2.3.7 Available keyboard shortcut list

To activate the commands of **Debug,** the options can be selected the menu ***Debug*** or typed directly on the keyboard.

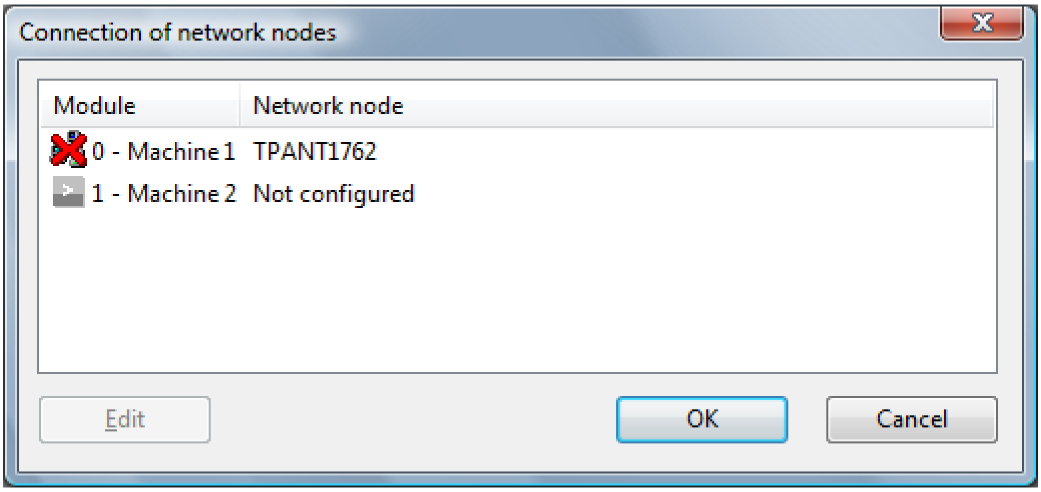
The keyboard shortcuts are as follows:

Key	**Description**
Ctrl+F5	opens the dialog window showing the list of the tasks in execution
Ctrl+Shift+F5	opens the dialog window showing the list of all the tasks
Ctrl+B	opens the dialog window to insert or cancel the breakpoints
Ctrl+F9	inserts or eliminates the breakpoints on the row where the cursor is placed
Ctrl+K	opens the dialog box to display the list of the functions called, but not yet returned
Shift+F9	opens a dialog window to display the content of a variable
F8	executes the instruction If this is a function, it enters the function
Shift+F7	executes all the instructions of the function
F10	executes the instruction If this is a function, it executes it without entering
F7	executes all the instructions until the instruction where the cursor is placed. The cursor should be placed on an instruction within a function
Alt+Interr	interrupts the execution of the code at the last executed instruction
F5	resume the code execution after an interruption
Shift+F5	ends the current task and executes it again
Alt+F5	ends the debug

2.4 Control initialization

2.4.1 Network Connections

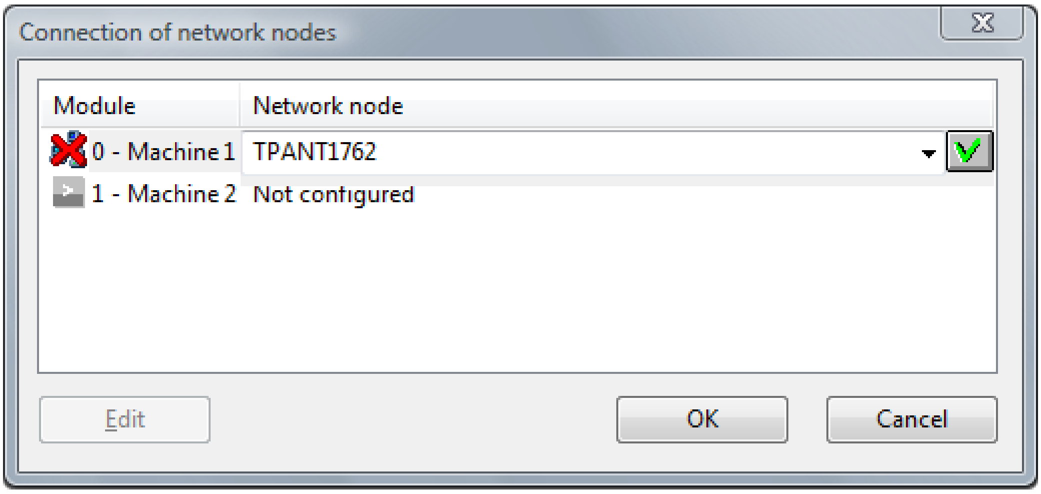
The profile machining of Albatros is protected by a USB hardware key, configured by T.P.A. S.p.A.
This command can be selected from the menu ***Cnc->Network Connections.*** It displays the state of the remote modules connected to the system. If a module is not connected, the symbol with which it is indicated is marked with a red cross.
Each module has two fields. The first one is the name of the associated module and the second one is the name of the network station. Usually the name of the network station begins with the fixed characters "TPANT" or "TPACE" followed by the serial number of the remote module.



Module	Network node
0 - Machine 1	TPANT1762
1 - Machine 2	Not configured

Remote modules connection

Assigning a network node to a logical module
To assign a network node to a module, position the mouse pointer on the text "Not configured" or click on the button **[Edit]**. A few seconds later a window containing the list of available remote modules in the network will appear (each remote module must be switched on and it must have received an IP address correctly)



Module	Network node
0 - Machine 1	TPANT1762
1 - Machine 2	Not configured

Assigning a remote module

Now, select the network node you want to connect to the logical module and confirm your choice by pressing the ✓ button.

Notice that this operation can be carried out at a "Service" password level, without having to access Albatros's System configuration for which a "Manufacturer" password level is required.
However, the module must be configured as "remote ALBRTX" in System configuration, beforehand.

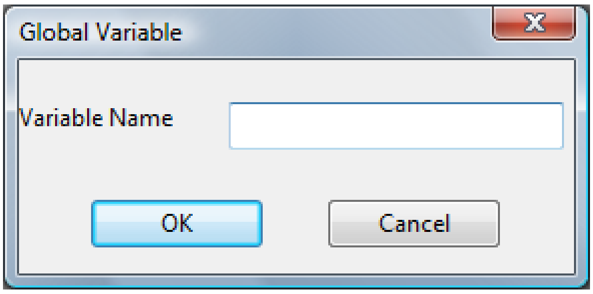
2.4.2 Hardware Diagnostic

This command can be selected from the menu ***Cnc->Hardware Diagnostic.***
Hardware Diagnostic displays the list and the state of configured modules, of axis cards and of the remotes belonging to them, as defined in hardware configuration. If the symbol of a card or of a remote is marked with a red X, it can either mean that this item was not found among the hardware in the control panel or that it was not possible to initialize it correctly.
If an item is marked with a yellow question mark, it means the system has detected a card or remote, but it does not match the type defined in configuration.

2.5 Test

2.5.1 Print global on disk

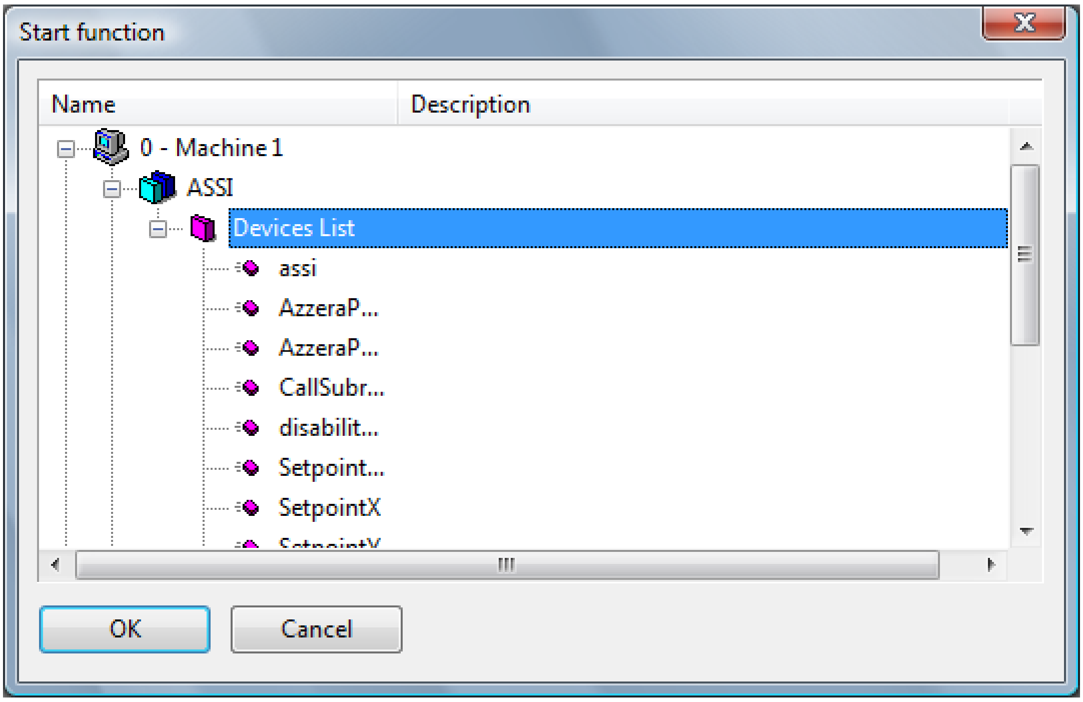
This command can be selected from the menu. It saves the content of a global variable on disk as a formatted text file. The file's name is *variablename*.txt and the file is saved in the *Report* folder.
This operation can only be performed if the read access level of the global variable is compatible with the current access level.



Saving a global variable

2.5.2 Start single function

This command can be selected from the menu ***Test->Start function.***
It executes a function independently of the rest of the system, creating a new task. The task begins its execution from the selected function, from which it will take its name.
Only the functions without input parameters and whose read access level is compatible with the current access level can be executed. If the executed function is the main function of the inter-group, all the autorun tasks will also be executed after.



Selecting a function to be executed manually

2.5.3 Message Import and Export

Group messages, assigned by means of the GPL DEFMSG instruction, can be stored in a text file to be modified and later re-introduced into the GPL code. This function is useful, for example, when you need to translate messages or create an archive of used DEFMSG instructions.
To import or export group messages, all the GPL code must be compiled without mistakes. Otherwise, the user would be prompted with a message saying "Not all the GPL code is compiled".
Group messages belonging to encrypted files cannot be exported or imported (See Chapter ***Development tools->Editor GPL->Cryptography***). Therefore, the user is not authorised to decipher (or decrypt) these group messages into plain text.

Export Group Messages
This command can be selected from the menu ***Test->Export group messages.***
A dialog box prompts you to enter the name of the text file where to store group messages. The default name is MSGEXP.TXT and it is saved in the folder defined in tpa.ini at the *dirReport* item.

Import Group Messages
This command can be selected from the menu ***Test->Import group messages.***
A dialog box prompts the name of the text file from which you can retrieve group messages to be introduced into the GPL code. The default name is MSGEXP.TXT. It is saved in the folder defined in tpa.ini at the *dirReport* item. Only the messages which have already been defined in the GPL code can be imported. The GPL text cannot be modified if there is at least one DEFMSG instruction following an IFDEF instruction.
While importing group messages, errors can be detected when:

- among the texts of a particular group message, the language identifier code is present more than once
- a text is empty (that is: "")
- the name of a group or a library is defined more than once.

At the end of the import process all modules, containing modified groups or libraries, are compiled.

File Format
The file is in text format. The keywords are GROUP, LIBRARY, AUXLANG and each language is identified by the relevant three-letter name.
Here is an example of how the file can be written:

```
;Complete list of messages

GROUP Main:                    ;Main group of any module
```

```
     MSG_BASE            ITA   "Italian translation"
                         DEU "German translation"
                         ENG "English translation"
                         ESP "Spanish translation"
                         FRA "French translation"

 GROUP 1.Main:                ;Main group of module 1
  MSGERR "Error of the only Main group of module 1"
LIBRARY Calculations:
TOOCOMPLEX                ITA "Troppo complesso" ENG "Too complex"
ERROR                     ITA "Errore generico"  ENG "Generic error"
BADARG                    ITA "Argomento errato" ENG "Bad argument"
```

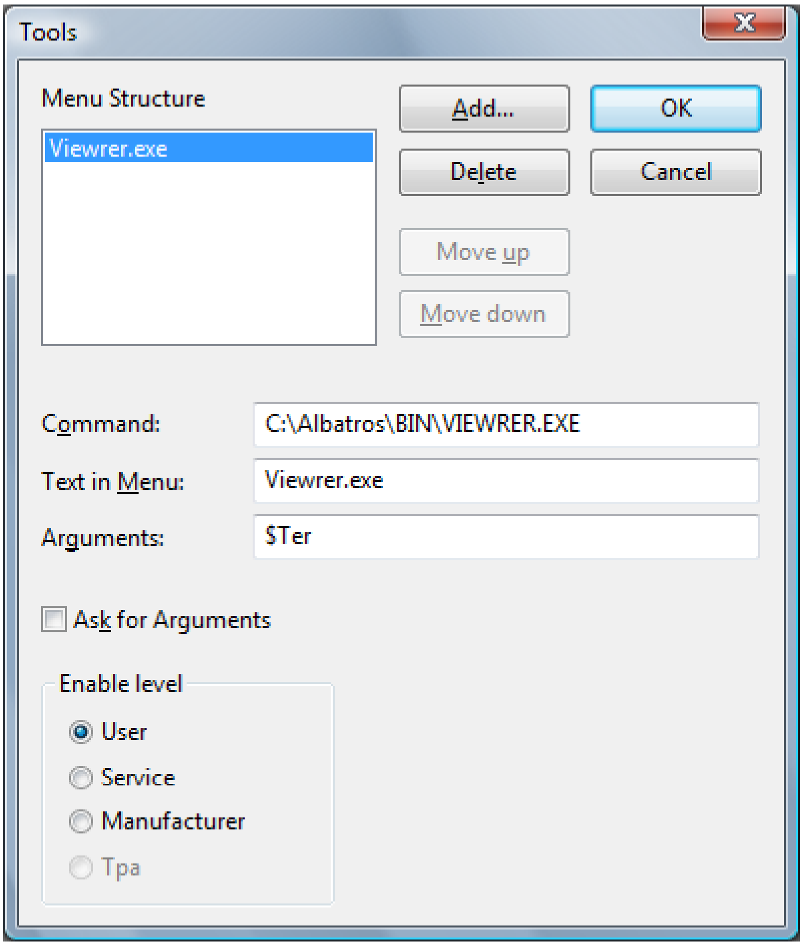
GROUP: it assigns the name of the group to which messages belong ("GROUP Main:"). If groups with the same name already exist in different modules, messages are imported in all groups. If you like that a few messages are imported only in one group of a given module, you need to put the module number and a “.” (point) before the group name ("GROUP 1.Main:").

- **LIBRARY**: it assigns the name of the library to which messages belong ("LIBRARY Calculations:").
- **DEFMSG Description**: it assigns the DEFMSG parameters: label (mnemonic name of the message to be displayed), language prefix (language in which the message is written: one of the 5 basic languages), message string (message to be displayed. It should be placed between quotation marks (""))
- **AUXLANG**: it assigns the name of the additional language used, while importing a group message, to enter an “additional” message into the GPL code, when the desired language is not included in the five main languages. It should be specified before the first GROUP or LIBRARY. ("AUXLANG: SQI")

2.6 Tools

2.6.1 Customise...

This command can be selected from the menu ***Tools->Customise.***
It allows you to set a maximum of 10 programs whose execution can be started by Albatros's ***Tools*** menu.



Menu Structure: Viewrer.exe

Command: C:\Albatros\BIN\VIEWRER.EXE

Text in Menu: Viewrer.exe

Arguments: $Ter

□ Ask for Arguments

Enable level

● User

○ Service

○ Manufacturer

○ Tpa

Configuration of the Tools menu

Menu Structure: lists the programs displayed in the ***Tools*** menu.

Command: name of the program to be executed. The folder in which the program is stored may also be indicated, especially if it is not the same folder from which Albatros is executed or from the folders whose operating system looks for the executable files (variable of PATH windows environment).

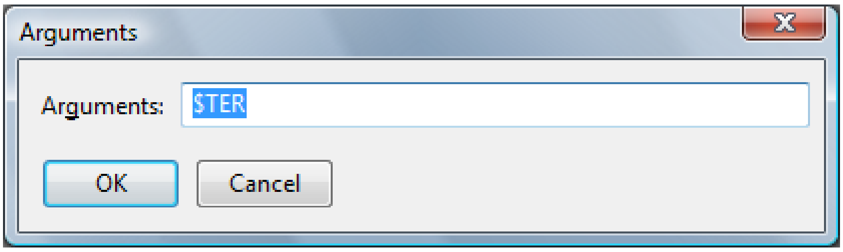
Text in Menu: the name appearing in the ***Tools*** menu to identify the executable program.

Arguments: any combination of command line arguments needed by the program for correct execution. It is possible to insert dynamic subjects. For exemple by using the string $TER during ViewRER execution report file of current month open.

Here is subjects list:

$File	Complete Path name of current file.
$FileName	File name and extension of current file.
$FileDir	Disc and folder of current file.
$Ter	Complete Path name of report file of errors of current month.
$DirModule	Disc and folder containing MODx of current file.
$Module	Module number of current file.
$Bin	Disc and folder containing Albatros executables.
$TpaIni	Complete Path name of initialization file TPA.INI
$ReqDirModule	Path (disk and folders) of Albatros module. If several modules are configured, the module dialog box opens. Example: $ReqDirModule\config\canbus.def corresponds to the path c:\albatros\bin\mod.1\config\canbus.def if the second module is selected.
$ReqModule	Albatros module number If several modules are configured, the dialog box of the module number opens.

Ask for Arguments: if selected, whenever program execution is requested, a dialog window appears to allow you to introduce different arguments from the ones set in the Arguments field. These can vary according to the launch mode of the program.



Arguments: $TER

It specifies the program-start arguments

Enable level: it sets the display level of the program in the ***Tools*** menu. Albatros's test programs and data modification programs are normally given a manufacturer level. Machining editing programs are assigned a user level.

Certain fields can be edited using the **[Add]** button. This opens the **Add Tool** dialog window for the selection of the program to be executed. The allowed executable files are the following: .EXE, .COM, .PIF, .BAT.
When the dialog window is closed, after confirming the data, the program is inserted in the **Menu Structure** window and the name of the program and its folder, in the **Command** row.
The other buttons provided are **[Delete], [Move Up], [Move down]** which are used respectively to delete a program and order the list of programs.

2.7 Browser

2.7.1 The browser

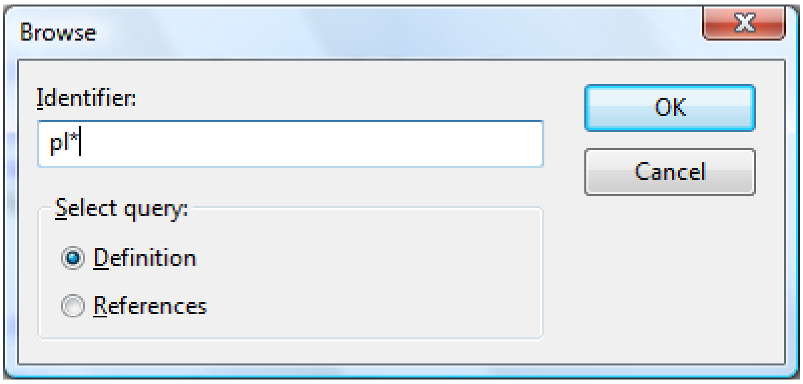
Albatros's browser function uses the information generated by the compiler to create a database for the rapid search of symbols defined in the functions.
This function can only be activated at manufacturer or higher access levels. To select the commands, use the ***Debug*** menu.
The browser enables to:

- position the cursor in the line where a function, or a module, group or library variable or a module or group constant is first defined (from the menu ***Debug->Go to definition***)
- position the cursor in the lines where a function, a device, a module or group variable or a GPL instruction (except for FCALL and FRET instructions) is mentioned. (from the menu ***Debug->Go to reference***, to display the previous reference or the next one select from the menu respectively the options ***Debug->Previous*** o ***Debug->Next***)

Group variables can only be managed from the edit window of the group they belong to.
To update the browser when switching to a new version, it is advisable to save the global variables first, and then execute the command ***File->Compile All.***
When editing the functions, the link between text and symbols is lost. The link is reestablished in the filing stage.

2.7.2 Identifier Search

This command can be selected from the menu ***Debug->Source browser.*** The identifier search opens a dialog window that allows you to insert the name of the symbol to be found in the GPL code. According to the selected **Type of search**, this function will find either the definition or the first reference to the symbol.



Browse

Identifier: pl*

Select query:

◉ Definition

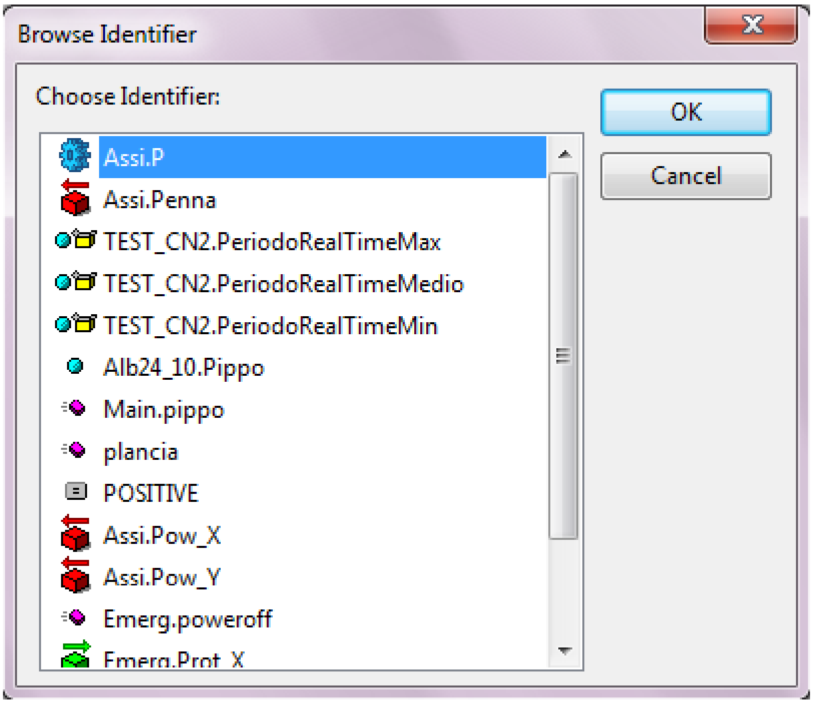
○ References

Identifier search window

The inserted name can have the following characteristics:

- if it contains no "." (period) character: the name is searched for in all the function files.
- if it contains only one "." (period) character: the name preceding the period is identified as the name of the group, and the symbol will only be looked for in that group. For example, if a VisError function has been defined both in the MAIN group and in the AXES group, when a search is called for AXES.VisError, the cursor will go to the first row of the VisError function in the AXES group.
- if it contains two "." (period) characters: the name preceding the first period is identified as the name of the group and the one preceding the second period is identified as the name of the subgroup. The symbol will only be searched for in that subgroup.
- if it ends with an "*" (asterisk) character the search will include all the symbols beginning with the characters preceding the asterisk.

In case of ambiguity in the search for a symbol, a dialog window is opened displaying all the symbols with the requested name. From this window it is possible to select the required symbol.



Browse Identifier

Choose Identifier:

Assi.P
Assi.Penna
TEST_CN2.PeriodoRealTimeMax
TEST_CN2.PeriodoRealTimeMedio
TEST_CN2.PeriodoRealTimeMin
Alb24_10.Pippo
Main.pippo
plancia
POSITIVE
Assi.Pow_X
Assi.Pow_Y
Emerg.poweroff
Emerg.Prot_X

Identifier selection window

Below is a description of the special symbols used in the list for the identifier selection.

Symbol	**Description**
🔨	GPL instruction
⊟	module or group or library constant
●	module or group variable

library variable

library vector

library matrix

library function

group message

label

local variable

local vector

local matrix

single parameter

array parameter

matrix parameter

2.7.3 Available keyboard shortcut list

To enable the Browser commands, select the menu items ***Debug*** or type directly on the keyboard.
The keyboard shortcuts are as follows:

Key	**Description**
F2	positions the cursor on the line where the selected symbol is defined. If the browser data-base contains several symbols with the requested name, a dialog window opens to allow the user to select the required symbol.
Shift+F2	positions the cursor on the first reference to the selected symbol. In case of ambiguity a dialog window opens to allow the user to select the required symbol.
Ctrl+F2	opens a dialog window for the selection of the required symbol.
Ctrl+'+' or Ctrl+PgUp	positions the cursor on the following reference (use the "+" on the numeric pad)
Ctrl+'-' or Ctrl+PgDown	positions the cursor on the previous reference (use the "-" on the numeric pad)

3 GPL Language

3.1 Basic Feature

3.1.1 Conventions and terminology

Basic terms

ARGUMENT
One of the arguments of the instructions; it can be defined as *constant*, *variable*, or *parameter*, depending on the kind of instruction; if between square brackets (**[ ]**) it means that it may be omitted, implying that the instruction can be executed in a different way.

KEYWORD
An argument to be chosen among the arguments with a predetermined value, normally written in capital letters; the list of keywords is provided in a specific help page.

PARAMETER
The argument of an instruction which is not defined within the instruction, but is passed to the function, precisely as a parameter, when the function is executed; in certain cases it is also called *parameterised argument*.

CONSTANT
A fixed argument defined by means of the CONST metacontrol or an argument which is rigidly fixed within the instruction.

VARIABLE
An argument defined as machine or group global variable or defined by a LOCAL instruction, which can be organised as simple variable, vector or matrix. See variables.

CONFIGURATION PARAMETER
An argument defined in configuration, such as the parameters of an axis, for example.

Most frequent arguments in instruction descriptions

The list below contains the terms relating to arguments which are frequently used in GPL instruction syntax. Each one is followed by a brief description. In cases in which an argument can assume a different value from the one described below, its description continues in the *arguments* section of the instruction's help page.

inputname	name of digital input device
outputname	name of digital output device
flagname	name of flag switch or flag bit device
portname	name of input port, output port or flag port device
timername	name of timer device
countername	name of counter device
functionname	name of a function (also valid as device parameter in the case of ERRSYS.)
subprogramname	name of a subprogram, it is the equivalent of *label*, to which we refer to for explanations; to call a subprogram use the instruction "CALL subprogramnameme".
axis	name of an axis
constant	a character, an integer or double number, or a keyword
value	constant or variable (the *type* depends on the instruction)
variable	name of: variable, vector element or matrix element
variabledevice	name of *device parameter*
matrix	name of a matrix
vector	name of a vector
label	name of the jump label or name of a subprogram.
state	logic state, options: ON or OFF, or 1 or 0
timeout	amount of time within which something has to happen, or a delay time (constant or variable)
position	coordinates of the position (double constant or double variable)
radius	value of the radius (double constant or double variable)

angle	value of the angle (double constant or double variable)
numrev	number of revolutions (double constant or double variable)
speed	value of speed (float constant or float variable)
direction	clock or anti clockwise rotation (variable or constant: CW o CCW)
operand	(constant o variabile o devicename)
result	result of the operation (variable or devicename)
devicename	name of any type of device (or device parameter)
constantstr	sequence of characters in inverted commas (ex. "string")
variablestr	the name of a character vector, namely a string
operator	comparison operators: > (greater than) = (equal to) < (less than) they can also be used in combination, for ex. >= (meaning: greater or equal to)
type	type of constant or variable: "char" (8 bit), "integer" (32 bit), "float" (32 bit), "double" (64 bit), "string"
device parameter	is a variable that stands for a device. The devices are defined in Configuration.

Main terms used for axes

theoretical position (or target)
Current "theoretical" position set, second by second, by the numerical control on the basis of the algorithm of speed profile generation.

real position
Real position of the axis as detected by the position transducer. The difference between the real position and the theoretical position is known as "tracking error" or "loop error".

final position
It corresponds to the programmed arrival position of a movement. The calculation algorithm of the speed profile enables the theoretical position to reach exactly the final value.

arrival position window
Programmable interval whose central point corresponds to the final theoretical position: when the real position enters this area, the movement is considered concluded.

arrival position big window
Position arrival window multiplied by a factor to be set by means of the instruction SETBIGWINFACTOR.

loop error
The difference, second after second, between the theoretical position and the real position of an axis: it is usually proportional to translation speed and inversely proportional to the "proportional loop gain".

proportional [loop] gain
Axis regulation parameter, programmable: it determines the ratio between current speed and relative loop error.

feed forward
Axis regulation parameter, programmable: it determines a direct contribution (proportional to programmed speed) injected on the drive speed control. It allows you to reduce, at equal speed and equal proportional gain, the value of the loop error.

feed rate override
Percentage of programmed speed. This parameter allows you to reduce execution speed, compared to programmed speed, by a percentage ranging between 0% and 100%.

tolerance
Move value according to which the axis moves away from the original trajectory in a multi-axis interpolation between two consecutive blocs of displacement.

backlash
Space between the cogs of a couple of gears.

3.1.2 Introduction to GPL language

GPL language (General Purpose Language) is the language used to create functions in the Albatros system.

Although its structure, for some aspects, is similar to BASIC, it is characterised by a large number of device control instructions.
The language is composed of more than 200 instructions, called *instruction*, which have been divided into groups of instructions with similar functions, for your convenience.
Moreover, the language is multitasking, allowing the execution of various tasks at the same time.

Typical Syntax of GPL instructions
GPL instructions all have a similar structure, corresponding to the following pattern:

```
instructionname parameter-1, parameter-2, ..... parameter-N
```

The number of parameters depends on the instruction and the contest in which it is used, the absolute maximum paremeters number for a function or an instruction is 120. In certain cases the instruction may not contain any parameters at all.

The smallest block of GPL code is the function.

Dividing the code into groups
The GPL code is subdivided into blocks that reflect the logic subdivision of the machine into groups. This means that each group has a corresponding file containing its code. To these files, containing the code of the groups present in the machine, we must add the file containing the global variables and constants which are visible from any group's GPL code and the libraries. These contain code not related to machine configuration hence easily portable to other machines.

3.1.3 Variables

Variables are information containers which in the GPL language are used to store all the values necessary for program functioning.
Variables are characterised by a "type" that indicates the kind of information they contain. Moreover each variable has a specific visibility which determines which code groups or subgroups can operate (read or write) on it.

Type of data

SIMPLE OR SCALAR DATA
GPL supports both simple and aggregate data. The types of simple data are similar to the ones used in most programming languages:

Char
Is an integer with sign ranging between [-128 ; +127] and its length is 1 byte.
To declare a Char variable, the following syntax is used:

```
VariableName as char
```

Integer
Is an integer with sign ranging between [-2147483647 ; +2147483647] and its length is 4 byte (it corresponds to the long type in C).
To declare an Integer variable, the following syntax is used:

```
VariableName as integer
```

Float
Is a floating point number ranging between [-3,402823 E+38 ; -1,401298 E-45] and [+1,401298 E-45 ; +3,402823 E+38], its length is 4 byte (it is usually used to indicate speed).
To declare a Float variable, the following syntax is used:

```
VariableName as float
```

Double

Is a floating point number ranging between [-1,79769313486231 E+308 ; -4,94065645841247 E-324] and [4,94065645841247 E-324 ; 1,79769313486231 E+308], its length is 8 byte (it is usually used to indicate positions)
To declare a Double variable, the following syntax is used:

```
VariableName as double
```

These types of data can be used together in one expression. The GPL converts them automatically without giving any warning messages. For this reason, when using different types of data in the same expression, it is advisable to check that no information has gone lost.
In certain situations conversion is not allowed. In this case the compiler usually sends an alert message or a system error occurs.

AGGREGATE DATA

Array

It is a group of simple variables, all of the same type, obtained by associating an index to the name of the variable. The index must be enclosed in square brackets. If the array is called, for example, "parameters", the first item of the group will be called "parameters[1]", the second "parameters[2]", and so on.
The array has a fixed number of items which must be determined in the declaration. A typical array declaration uses the following syntax:

```
parameters[10] as integer
```

Where *parameters[10]* indicates that the name of the array is "parameters" and that it's composed by 10 items; *as integer* indicates the type of simple data used for the array's individual elements, which in this case is an integer.
The arrays can be made up of simple data or strings.
An array can have a maximum of 262144 elements.
Vectors can be directly initialized in the GPL code, at the time of their declaration. GPL syntax can be:

```
[READONLY] vector[numberofrows] as integer  = 1,2,3,4
[READONLY] vector[numberofrows] as string  = "one","two","three","four"
```

Matrixes

Matrixes are bidimensional arrays, that is, variables with two indexes. A matrix can be visualized as a table divided into rows and columns. To indicate a cell on the table, we can indicate in which row and which column it is. The first index indicates the number of the row and the second the number of the column.
Unlike arrays, matrixes can contain different types of data, but with the following restriction: we may use a different type of simple data for each column but it is not possible to vary within the column. For example we can define a matrix in which the first column is integer type and the second is float type. However we can not have a matrix where the first row is occupied by an integer and a float and the second by a char and a double. In the rows, the elements must all be composed by the same type of data.
The declaration of a matrix can be written using the following syntax:

```
offset[10] as double double double

dim_part[50] as float:length float:width float:thickness
```

In the second type of declaration a label or symbolic name is given to each column. The symbolic names of the columns are very useful when working with large matrixes, as in this kind of situation it s difficult to remember the values memorised inside each column of the matrix. The symbolic name allows us to identify immediately the type of data we are working with. For ex. " "Offset[1][3]" is not as clear as "Offset[1].axis_X".
Matrixes can only contain simple data. For example, it is not possible to create matrixes containing strings. The maximum number of rows in a matrix is 262144.
Matrices can be directly initialized in the GPL code, at the time of their declaration. GPL syntax can be:

```
[READONLY] matrixname[numberofrows] as double double integer double = _
   1.1, 2.2, 3, 0.1 _
   1.2, 3.4, 5, 0.1 _
```

2.1, 5.6, 6, 0.1

Strings
Strings are groups of characters, that is char data. However, because they represent legible text, they are treated in a special way.
A string is very similar to a char array. The main difference is given by the presence of a terminating character, which is automatically added at the end of the string. The GPL also provides some macros which allow you to manipulate the strings.
Usually strings are used to write messages, which the user can read on the screen or in a report file.
To declare a String variable, the following syntax is usually used:

```
VariableName as String
```

To declare a String variable, the following syntaxes may be used:

```
VariableName as String

VariableName[20] as String
```

In the first declaration the string assumes a default size of 256 characters. In the second case a maximum string size is defined.

Data conversion

In all mathematical expressions, but EXPR instruction, the types of data of the operands are converted according to the type of data of the result variable and then the operation is executed. It is important to pay attention to the declaration of types of data, because they can influence the result. Following table is an example of how the results based on the type of data given may change:

DIV	Operand 1(Integer)	Operand 2(Double)	Result (char)
	3	5.0	0
	5	1.9	5
	1200	107.2	Undefined
	1200	250.0	Undefined

DIV	Operand 1(Double)	Operand 2(Double)	Result (Double)
	3	5.0	0.6
	5	1.9	2.631
	1200	107.2	11.194
	1200	250.0	4.8

In the EXPR instruction, if the operands are not of the same type, an automatic conversion is carried out and the type of the result of the operation is the same as the greater one of the two results, according the following rule:

- char <integer
- float < double
- char or integer < float or double.
-

After resolving the expression, the result is converted according to the type of the result variable.

EXPR	**Operand 1(Double)**	**+**	**Operand 2(Integer)**	**/**	**Operand 3(Float)**	**Result (Integer)**
	900.0	+	100	/	400.0	900

EXPR	**Operand 1(Double)**	**+**	**Operand 2(Integer)**	**/**	**Operando 3 (Float)**	**Result (Double)**
	900.0	+	100	/	400.0	900.25

Declaration and Visibility of the variables

Variables and constants can only be declared in specific parts of the GPL code.
We can classify as variables:

- Module globals
- Group globals
- Locals (variables only)
- Library globals

A maximum of 2048 variables (module and group) can be declared.

It is possible to define some *modifiers* that assign additional characteristics to the variables.

Module global variables

Module global variables are grouped in a special file which is accessed by selecting the heading ***Menu->File->Open Global Variables.***
The declaration is performed, as shown in previous paragraphs, by specifying the name of the variable, followed by the keyword "AS", followed by the type of data (or types of data in the case of matrixes).
These variables are visible directly from the code of all the groups.

Group global variables

Group global variables are defined at the beginning of the group code. They must be declared before the GPL functions.
These variables are directly visible from the integer code inside the group. Moreover it is possible to extend the visibility of these variables outside the group by declaring them as "Public" variables.
Public variables are not directly accessible from outside the group. To access them, we have to use their name preceded by the name of the group they belong to. For example, if we want to modify the "offset" public variable, belonging to the "axes" group, from the code of the "main" group, we will write "SETVAL 10 axes.offset".
To declare a group global variable, the same syntax used for module global variables is used. The main difference lies in the definition of public variables. To define one or more public or private variables use the labels "Public" and "Private". For example:

```
Public:
  offset as double
  speed as float
Private:
  tool as integer
```

Local Variables

Local variables are declared in the body of a function. They must be declared before any other instruction, except for the declaration of the function's parameters.
Local variables are only accessible from inside the function.
These variables are created with a 0 value (the necessary memory is allotted) only at the beginning of function execution and are destroyed (the memory is released) at the end of execution. Global variables, on the other hand, are created when the module is initialized and are always visible in "Diagnostic".
The declaration of a local variable uses the syntax we have already seen, but is preceded by the keyword "LOCAL".
For example:

```
Function processing
local    position_centre_ as double
movabs X,position_centre
fret
```

Library global variables

Library global variables are declared in GPL code libraries. They are similar to group global variables.

Modifiers

Modifiers: READONLY

Module and group global variables can be declared as READONLY.
A readonly variable is a variable whose value can not be modified by the GPL code, although it can

be modified from "outside", that is by Albatros's technological parameters file.
The technological parameters file is a database which stores the values that characterize the machine but could vary in the long term if the machine were modified or in case of extraordinary maintenance. This data is normally inserted in a GPL matrix during control initialization.
An example of this type of information are the machining area offsets or the dimensions and technological parameters of the tools.
By declaring these variables as readonly we avoid accidental modifications of the information which shouldn't vary during normal machine functioning.
The maximum size of a readonly variable is 128 Kbyte.
To declare a readonly variable, the following syntax is used:

```
readonly VariableName as type
```

Modifiers: NONVOLATILE
Variables declared as NONVOLATILE class are memorized on the non volatile RAM (provided with batteries) instead of the normal RAM. Consequently the values stored in these variables are not lost when the numerical control is switched off.
For the declaration of a nonvolatile variable, the following syntax is used:

```
nonvolatile VariableName as type
```

For example:

```
nonvolatile OffsetArea[2] as double:offsetX double:offsetY double:offsetZ
```

Only group and machine global variables can be classified as "nonvolatile".
The maximum size of variables memorized on nonvolatile RAM is 15100 byte. The maximum size of a single non volatile matrix is 1024 byte.

Assigning a RANGE

When formulating a declaration it is possible to assign a range of values to the variable. However, at the moment, there is no control of limits observance in execution phase, except for a compiler control in the case of constant values (for ex. to initialize the variable).
Consequently, the main advantage is constituted by a sort of code auto documentation.
For the definition of ranges, the following syntax is used:

```
VariableName Range:minval..maxval AS type
```

For example:

```
ToolNumber Range:1..100 as integer
```

Writing and Reading Rights

Writing and reading rights allow you to specify the minimum access level to the system, necessary to display (read right) and modify (write right) its value.
The syntax used is:

```
VariableName Read=S Write=M AS type
```

The keywords used to specify the rights are:

• READ	reading
• WRITE	writing

The values which can be assigned are:

• U or USER	user
• S or SERVICE	service
• M or MANUFACTURER	manufacturer
• T or TPA	tpa

The values' defaults are:

• READ	reading for service (S or SERVICE)

- WRITE writing for manufacturer (M or MANUFACTURER) and tpa (T or TPA)

Constants

Constants

GPL uses four types of constants:

- Integer
- Double
- Char
- String

Char constants are declared by using inverted commas, as below:

```
Const COD = 'A'
```

String constants are declared by using inverted commas, as below:

```
Const MSG = "Start processing"
```

For Integer constants and Double constants the following syntax is used:

```
Const PI = 3.14
Const MSGBOX = 12
```

For Integer constants a binary and hexadecimal notation is allowed:

```
Const MASK = $11001001b          ; binary
Const MASK = $F5h                ; hexadecimal
```

Also group and library constants can be public or private.
The sintax is similar to variables' one.
Example:

```
Public:
   Const PI = 3.14
   Const MSGBOX = 12
Private:
   Const MASK = $11001001b
```

NOTE: Float constants do not exist. Decimal numbers must necessarily be declared as Double. In certain cases this might cause alert messages from the compiler (when optimized GPL macros are used for Float types).

The constants can be defined as the result of calculation expressions, with the following syntax:

```
Const a = 10
Const b = 20
Const c = a + b
```

Permitted operators are the same as those used in the EXPR instruction.

3.1.4 Predefined constants

The GPL language has some predefined constants, which can be used directly without having to define them.
The predefined constants and their respective values are:

ON	**1**
OFF	**0**
UP	**+1**
DOWN	**-1**
POSITIVE	**+1**
NEGATIVE	**-1**
CW	**1**
CCW	**0**

TRUE	**1**
FALSE	**0**
NOWAIT	**0**
WAIT	**1**
WAITACK	**2**
STORE	**1**
NOSTORE	**0**
NOPLACE	**0**
COM1	**0**
COM2	**1**
COM3	**2**
COM4	**3**
COM5	**4**
COM6	**5**
COM7	**6**
COM8	**7**
NOPARITY	**0**
ODDPARITY	**1**
EVENPARITY	**2**

3.1.5 Keywords

Keywords are identifiers with a specific function and can not be used in any other way.

Available keywords are:

All the names of GPL instructions See the "Instructions" part of the manual for the description of all GPL instructions

All kinds of data See Variables

Device parameters See Device parameters

EXIST Used in IFDEF instructions to verify the existence of a group. See IFDEF instruction

NOTEXIST Used in IFDEF instructions to verify the non existence of a group. See IFDEF instruction

LINKED used in the IFDEF instruction to enable the compilation of code blocks, if the device is connected in virtual-physical. See IFDEF instruction.

UNLINKED used in the IFDEF instruction to enable the compilation of block codes, if the device is not connected in vitual-physical See IFDEF instruction.

_ID_MODULE Used in the IFDEF instruction to verify the current module number. See the instruction IFDEF

_REMOTE_MODULE Used in the IFDEF instruction to verify if the connected module is a remote module (value=1). See instruction IFDEF

_VER_MAJOR Used in IFDEF instruction to verify the main version number of Albatros. See instruction IFDEF

_VER_MINOR Used in the IFDEF instruction to verify the secondary version number of Albatros. See instruction IFDEF

_VER_REVISION Used in the IFDEF instruction to verify the revision number of Albatros. See instruction IFDEF

_VER_SP Used in the IFDEF instruction to verify the service pack of Albatros .See instruction IFDEF

_VER_FULL	Used in the IFDEF instruction to verify the service pack of Albatros. See instruction IFDEF
FUNCTION	Declaration of a function. See Functions
AS	Used for variable declarations. See Variables
PUBLIC	An attribute of functions. See Functions
AUTORUN	An attribute of functions. It indicates that the function runs automatically. See Functions
R= o READ	An attribute of functions or variables. It indicates the read access level. See Functions, Variables and Access rights
W=o WRITE	An attribute of functions or variables. It indicates the write access level. See Functions, Variables and Access rights
CONST	It allows you to assign a significant name, called symbolic constant, instead of a number, character or string. See Variables
READONLY	An attribute of global variables. See Variables
NONVOLATILE	An attribute of global variables. See Variables
PRIVATE	An attribute of functions. See Functions
RANGE	Used for the definition of an interval of values for variables. See Variables
USER	An attribute of functions or variables. It indicates the type of access. In this case user. See Functions or Variables
SERVICE	An attribute of functions or variables. It indicates the type of access. In this case service. See Functions or Variables
MANUFACTURER	An attribute of functions or variables. It indicates the type of access. In this case manufacturer. See Functions or Variables
TPA	An attribute of functions or variables. It indicates the type of access. In this case TPA. See Functions or Variables

3.1.6 Functions

Functions are the smallest block of GPL code. GPL instructions can not be inserted in a file in sequence, they have to be grouped in functions.

As far as the compiler is concerned, a function is any block of GPL code beginning with a line whose first word is FUNCTION. However, there is no keyword indicating the end of the text of a function: the function ends with the line preceding the beginning of another function or with the end of the file containing the functions.

The syntax used to define a function is:

```
FUNCTION    FunctionName  Attributes
   Parameters
   Local Variables
   List of GPL instructions
```

A function is also a special type of Albatros device. As a device, it is characterised by a series of properties common to all devices: a univocal name (untranslatable), a descriptive name (which can be translated, although it can not be set in GPL text), a logic address, a visibility indicator (whether the device is public or not), an access rights for reading and an access level for writing (see next paragraph).

Access rights

Because functions are a special kind of device, they are subject to access rights like all other devices. Access rights allow you to specify the minimum access level to the system necessary to allow visibility (read right) and execution (write right).
The syntax used is the following:

```
Function      FunctionName    READ=S WRITE=M
```

The rights are identified by the keywords READ (reading) and WRITE (execution)
Assignable values, corresponding to the various access levels, are:

• U or USER	user
• S or SERVICE	service
• M or MANUFACTURER	manufacturer
• T or TPA	tpa

The values' defaults are:

- READ reading for service (S or SERVICE)
- WRITE writing for manufacturer (M or MANUFACTURER) and tpa (T or TPA)

Autorun Functions

Autorun functions are executed automatically when the machine is booted.
Autorun functions have a characteristic: they are restarted automatically after being closed down because of a system error.
The syntax used is the following:

```
Function      FunctionName    autorun
```

So it is sufficient to add the modifier "autorun" to the declaration of the function.

Public Functions

Normally a function can only be executed (called) by the code inside the group file. To make it possible for a function to be executed by the GPL code of a different group, it must be defined as **public**. The syntax used to define a public function is the following:

```
Function      FunctionName    public
```

So it is sufficient to add the modifier "public" to the declaration of the function.
Functions belonging to the intergroup are an exception, as they are always **public**.

Subgroup Functions

A function can be connected to a subgroup simply by putting the name of the subgroup in front of the name of the function. The subgroup and the function's name must be separated by a full stop ".". For example the following function belongs to "X" subgroup of the "Axes" group.

```
Function      X.homing
      local     vel as float
      movabs X,100
      waitstill  X
Fret
```

Asynchronous Functions

Asynchronous functions are automatically called by the numerical control when the event connected to the function takes place.
Three types of events are possible:

- Change of state of a digital input: instruction ONINPUT
- Change of state of a flag bit or flag switch: instruction ONFLAG
- System error: instruction ONERRSYS

When the event takes place, the function is called (not as autonomous task but in the context of the task in which the corresponding ON... instruction was executed) as implicit FCALL, as soon as the current instruction has terminated execution.
Typically, asynchronous functions are used to resolve emergency situations, and they must be extremely

fast. For this reason, these functions can't use just any GPL instruction; they use a subgroup which guarantees short execution times.

Functions with input parameters (parametric)
A function can have some parameters declared in input, without ever returning any values.
These parameters can be considered as special local variables whose value is initialized externally the moment the function is executed. The parameters are indicated with the keyword PARAM and use the same syntax used for local parameters. The parameters must be listed in the first lines of the body of the function, before any other instruction and before the local variables.
There are two ways the parameters can be passed:

- ***by value:*** all simple data types are passed by value, that is CHAR, INTEGER, FLOAT and DOUBLE. Passing by reference means that a copy of the original value is created. Changes made to the parameter only have an effect in the context of the function.
- ***by reference:*** aggregate data types are passed by reference, that is ARRAY, MATRIXES and STRINGS. Passing by reference means using the source variable; consequently the changes made to the parameter have an effect in the context of the calling function. This characteristic can be exploited to send return values back to the calling function.

Typically a function is sent in execution with the instruction FCALL. If the concerned function is a parametric function, the list of values to be given to the parameters must be specified after the name. In the following example we find a parametric function executing a perforation operation. The coordinates of the centre of the hole and feed speed of the Z-axis are passed to the function as parameters.

```
Function Perforation
    Param Qx as Double        ; position X of the centre of the hole
    Param Qy as Double
    Param vel as Float        ; feed speed

    Movabs        X, Qx, Y, Qy
    Waitstill     X,Y
    ....
    Fret
```

This function call, for example to make a hole in the position (12.5 , 25.7), with a feed speed of 3m per minute, could be written in the following way:

```
Fcall    Perforation 12.5, 25.7, 3.0
```

The parameters passed to the function must match in name and type, those declared in the call function. The execution of the call function restarts at the end of the called function.
It is also possible to declare a device as a function parameter. This enables to write general use functions, such as a homing function, to be used with all the axes in the machine:

```
Function HOMING PUBLIC
        param      axis as Axis
        movabs     axis,100
Fret

Function MAIN
        .....
        Axes.Homing x
Fret
```

The homing function belongs to the Axes group and is declared PUBLIC to allow it to be seen by the functions declared in other groups. The Main function calls the axes group homing function, specifying the axis which has to be moved as an input parameter.

3.1.7 Device parameters

Device type parameters are special variables which allow you to call a machine device.
This kind of data can be used **exclusively** in the declaration of function parameters. So it is not possible to declare variables of this type. The definition of names and other characteristics of the devices pertain to System Configuration.
The following table contains the type of Device and the relative keywords to be used for the declaration of the parameters.

Type	Keyword
Digital input	INPUTDIG
Digital output	OUTPUTDIG
Analog input	INPUTANALOG
Analog output	OUTPUTANALOG
Axis	AXIS
Timer	TIMER
Counter	COUNTER
Flag bit	FLAGBIT
Flag switch	FLAGSWITCH
Flag port	FLAGPORT
Input port	INPUTPORT
Output port	OUTPUTPORT
Input Nibble	INPUTNIBBLE
Output Nibble	OUTPUTNIBBLE
Function	FUNCTION (only for ONERRSYS)
Generic device	DEVICE (only for ONERRSYS)

Example of axis parameter declaration and use:

```
Function test
   Param axis as axis

   MovAbs     axis,100
   WaitStill  axis

Fret
```

3.1.8 Multitasking

As the system is multitasking, it is possible to have more than one GPL task in progress at the same time, and by task we intend the handling process of a logic entity (usually a group).

There are two types of task available: normal tasks and the most recent "Real-time Tasks".

Normal tasks
Multitasking is based on a cooperative algorithm based on priorities. This guarantees that all the tasks are executed cyclically, varying their priority. The scheduling algorithm ensures that one instruction is executed for each active task (running state). Every task has a priority set using the instruction SETPRIORITYLEVEL assigned to it. The priority is identified by a whole number between 0 (highest priority and 255 (lowest priority). For tasks with a priority of 0 (zero) an instruction is carried out every scheduling cycle, for tasks with a priority of 1 an instruction is executed every two scheduling cycles and so on up to tasks with a priority of 255 for an instruction is carried out every 256 scheduling cycles.

The execution of normal tasks is asynchronous with respect to the frequency of refresh of the axes. This means that there is no guarantee that a GPL function will be completed in the time span between two updates of the state of the axes.

A task is identified by the name of the GPL functio
n from which its execution starts.
The execution of a task can begin:

- automatically with the initialisation of the system: main intergroup function and autorun functions.
- following the execution of a STARTTASK execution.
- Following the triggering of Albatros in manual mode using the graphics interface.

Each task is characterised by an internal state:

RUNNING	The task is running
HOLD	The task is suspended
BREAK	The task has been interrupted by the debugger

Tasks are organised hierarchically in a tree structure. Each task is created by another, which means that if the mother task finishes, all the child tasks will also be terminated.
The maximum number of tasks in execution at the same time is 500.

It must be considered that an high number of running tasks implies a decrease in speed, at which every single task is performed.
If the application to be made is supposed to imply the use of a number of tasks higher than 200, the operator should use a proper hardware such as **Cn2128.**

Real-Time Tasks
Real-time tasks differ from the foregoing in that they are not subject to a scheduling procedure nor are they arranged by priority, but are executed completely with each update of the state of the axes (axes real-time).

It is absolutely necessary for the execution of these tasks to end by a set time because the execution of the GPL tasks described earlier remains on hold while the real-time tasks are being run.
The system runs checks on the execution time of real-time tasks and should these exceed the maximum time allowed the system generates an error.
It is therefore not advisable to create infinite cycles (e.g. using GOTO instructions) within these tasks; cycles, moreover, are not necessary given that the execution of the code starts again from the beginning with each axes real-time task.
In order to avoid excessively long execution times real-time task use is limited to some GPL instructions. The instructions whose use is not allowed are those that cannot be used on interrupt.

We advise using real-time tasks only for those activities that must of necessity be carried out synchronously with the update of the axis positions. For most control activities it is better to use normal tasks.

Real-time tasks are sent with the instruction STARTREALTIMETASK and can be interrupted with the instruction ENDREALTIMETASK. Up to 256 real-time tasks can be activated at the same time.
The tree structure is no longer applicable, so if the task creating a real-time task ends, the real-time task will still run.
The local variables declared in the realtime task are initialized only by the start of the task and then they maintain the value of the last run.

Real-time tasks are not characterized by states typical of normal tasks. A real-time task can be debugged, but when this happens the system automatically declasses the task to a "normal task" for the duration of the debug.

If a system error is detected in a real-time task, the task is declassed to a "normal task" and it is put on HOLD to allow it to be analysed with the debugger.

3.1.9 Communication

Communications between the GPL and the outside world occur in three different ways:

- SEND / RECEIVE
- Serial communication
- IPC

Send / Receive
The instructions SEND and RECEIVE implement a message-orientated communication mechanism.
The communication may occur within the same module (of little advantage), between different modules of a line or between the modules and the supervisor Albatros or with OLE applications.
The way it works is similar to e-mails; for every message there is an addressee, an identifier of the information sent (or requested), the information itself plus the service information. Albatros performs the collect and sorting function of the information and in some cases directly supplies the information requested.
This mode of communication is normally used to send working programmes between the supervisor and the control units, to synchronize the activity of the machines of a line and to interface with external applications (OLE server).

Serial communication
The GPL language supplies some instructions, for example, COMREAD and COMWRITE, that make it possible to send and receive data via the serial ports of the numerical control. It is thus possible to interface the control with external devices like inverters, terminals or PLCs. When correctly used these instructions make it possible to implement serial communication protocols like MODBUS-RTU etc.

IPC
IPC or Inter Process Communication is a communication mode between processes. In particular, this mode allows an area of memory to be defined which is shared by two or more processes and can be

used for data exchange. Compared with other methods of communication, for example OLE, IPC is less sophisticated as there is no check on the data transmitted but it is substantially quicker.
Typically it is used when big quantities of data have to be transmitted or in general when the performances supplied by the OLE interface Albatros are not adequate.
On the GPL side IPC communication is implemented using the instructions SENDIPC, WAITIPC and TESTIPC. The external processes, however, may refer to the APIs supplied by RTX (an application written in C or C++) or to the COM component **gplipc2.dll** supplied by TPA S.p.A. which simplifies use (in particular for MS Visual Basic applications).
In addition, IPC makes it possible to communicate with other real-time processes (developed with RTX) and thus to integrate Albatros with hardware produced by third parties in the system.
For further information contact T.P.A. S.p.A.

3.1.10 Variables used in programming

Most instructions have been written so as to allow operating with various types of variables (CHAR, INTEGER, FLOAT, DOUBLE). However, each instruction has been optimised for a specific variable; for the best performance during GPL code execution, we advise using the type of variable suggested in the description of each instruction. In general, we suggest following the table below, which associates the main quantities used in programming to the relative optimal types:

quantity	***type***
position	*double*
speed	*float*
time	*double*
counter	*integer*
value port / flag port	*integer*
value nibble / BCD	*integer*
timeout	*double*
analog input / output	*float*
director cosines	*double*
string control character	*char*
acceleration / deceleration	*integer*

3.1.11 Axes

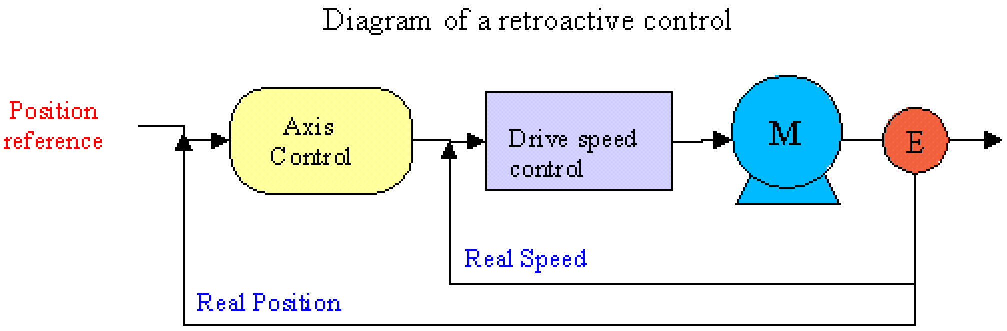
The term "axis" normally indicates an electromechanical system whose function is the controlled movement of a part of a tool machine.
Describing this system from the point of view of its components, we can subdivide them according to their technological characteristics.
The mechanical components are:

- frame
- guides
- bearings
- screws + ball screws

whose function is to contrast the forces involved, reduce friction, turn rotational motion into translation motion, etc.
The electric and electronic components are:

- motor
- end run switches
- encoder
- tachimetric dynamo

whose function is to provide the necessary power for movement and detect the state of the system.
These elements are connected so as to allow controlled execution of movements.



The function of the numerical control is to control the position and the movement of the axes.

Axis movement can be broken up into 5 phases:

Acceleration initial phase during which the speed of the axis is gradually increased, until it reaches programmed speed.

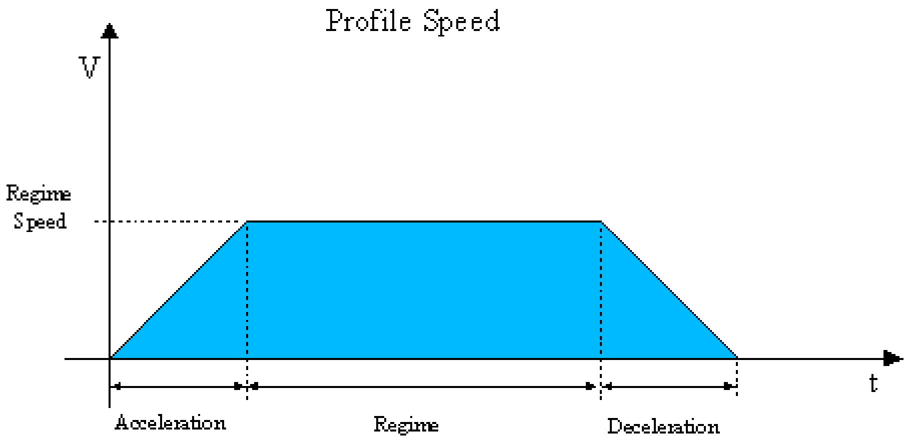
Regime intermediate phase during which the axis moves at constant speed (this phase may be omitted if the space to be covered is smaller than the space covered in acceleration and deceleration phases).

Deceleration phase during which the axis reduces its speed back to 0

Window pause, while the loop error is reduced to the value indicated in configuration as "arrival position window"

Position end of movement

At the end of the movement the axis will have to be positioned within an interval called "arrival position window" (that determines tolerance for axis positioning). If this is not done within 5 seconds of expected end of movement, the system generates a "movement not concluded" system error.



For each movement the numerical control calculates a speed profile like the one shown in the figure above. It then calculates the target positions by subdividing the speed profile in time intervals equivalent to axis refreshment time and calculating the area of each part. The area corresponds to the position increase which the axis has to reach in that space of time to comply with the above mentioned speed profile.

Axis control is implemented by means of a PID controller that "closes the position loop", meaning that, when the machine starts, it provides a speed reference calculated on the basis of the position that has to be reached (target position) and the real position read by the encoder. The difference between the real position and the target position is called ***Loop Error***.

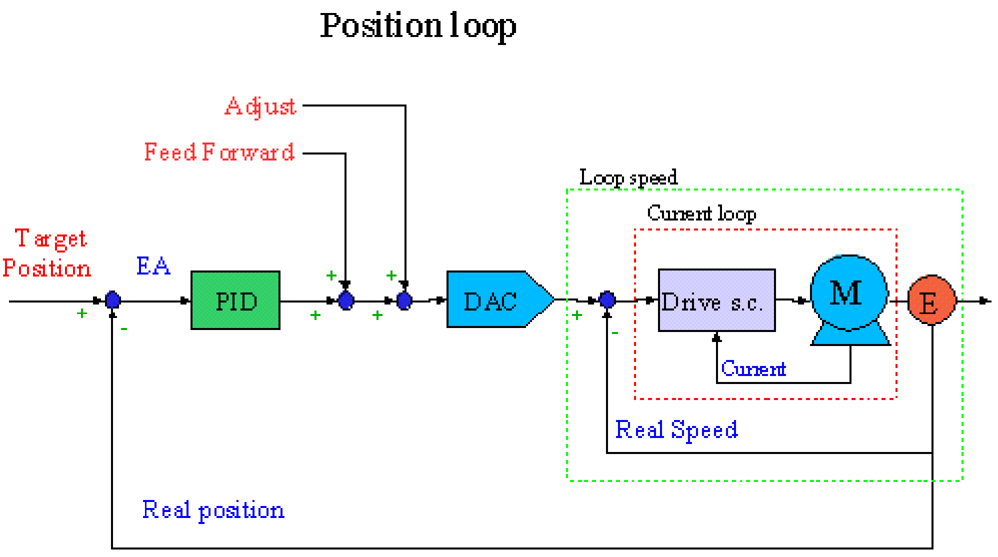


Diagram of Albatros axis control

3.1.12 Message handling in different languages

As said in the chapter describing the Composition of the System, Albatros supports the display of text messages in various languages.
This support is provided by a program independent of Albatros that manages the message files: Winmess.exe. Manufacturer is the minimum access level required to modify the language of the messages. Winmess reads the content of the language file and provides Albatros with the translation of the message in the selected language. It also handles language changes and memorizes the language selected by the user.

Text associated to Cycle Errors and Messages
Messages and Cycle errors are a special kind of text generated by the GPL code which are displayed by Albatros.
These are normally defined by the person who develops the GPS when writing the code itself. To simplify the programmer's work, the GPL editor allows you to insert the text of a message directly from Albatros, without having to use Winmess.
A second option for message handling in various languages is using the GPL DEFMSG instruction**.**

3.1.13 System Error Management

Whenever a system error occurs (See Chapter ***System Errors->Introduction to System Errors)*** the normal control behaviour is that of ending all tasks: the system error management allows you to avoid ending the tasks for which this function was enabled.
System errors generated by faults, stack underflow and stack overflow are directly managed by the relevant control without recalling the function of system error management: the task is placed in HOLD status.

Error Management Function
Within the GPL code, one or more functions should be defined to examine the system error and consequently to establish the most suitable actions to set the machine in safety conditions. The function to recall is passed as a parameter to the GPL ONERRSYS instructions. (See Chapter ***GPL Language->Instructions->Flux management->ONERRSYS).***

Whenever a system error occurs, the task which generated this error is placed in HOLD status. In case the autorun tasks generate system errors, they are relaunched only if the system error is not a FAULT. If the system error is generated without task number, the current task is palced in HOLD status.

3.1.14 Special functions

Axis movement customization

Albatros system graphical interface allows you to perform manual axis movements and provides a graphical tool for axis calibration.
Manual axis movement is performed by the manual movement control board, calibration may be performed by the calibration control board. Both can be accessed by the Diagnostic window and synoptic views.
In both cases axis movement is controlled by a set of GPL functions whose execution is hidden to the user.
The system has a predefined set of these functions which are adequate in most cases. Anyway in some cases may be necessary to customize the functions, for instance to define axes movement restrictions depending on to machine status or to manage auxiliary devices as drive brakes.

Customisation is performed by creation of two GPL function for each axis: one for the manual movements and one for the calibration. These functions are optional, if the system finds them uses them, otherwise standard ones are used. Furthermore a partial customization of the movement functions is possible.

Manual axis movement

The customized *manual movement* functions must respect the following rules:

- The function must belong to the same subgroup of the referred axis.
- Function name must be **MoveAx#*axis_name*** where axis_name will be changed to the axis name as defined in Configuration. For instance X axis function name will be: MoveAx#X .
- The function must provide the following parameters:

1. ***Required action***. May be an absolute position movement, an incremental movement, a stop etc. Actions are identified by an integer number, the GPL compiler provides a predefined constant for each action:

_MOVAXABS	absolute position movement
_MOVAXINC	incremental movement
_MOVAXSET	position setting
_MOVAXFREE	free status setting
_MOVAXNORMAL	normal status setting
_MOVAXEND	axis status reset after a movement (not used to stop the axis)

2. ***Result***. Needed by the system to know whether the required action may be performed by the customized function. If the required action is not supported, the corresponding standard function is used. So this is a return value that the customized function has to set, therefore it is defined as a "by reference" parameter (one element array).
3. ***Speed***. Meaningful only when the required action is a movement, it is the required movement speed.
4. ***Position***. Meaningful only for movement and position setting actions.

Custom axis movement function example:

```
Function MoveAx#X
   param action as integer
   param result[1] as integer
   param speed as float
   param position as double

   setval      1,result[1]

   select action
   case _MOVAXEND
         fcall EndMovement
```

```
      case _MOVAXABS
            fcall AbsMovement X, speed, position
      case _MOVAXINC
            fcall IncMovement X, speed, position
      case _MOVAXSET
            fcall PositionSet X, position
      case _MOVAXFREE
            fcall FreeAxis
      case _MOVAXNORMAL
            fcall NormalAxis
      case else
            call  Unknown
      endselect

      fret

   Unknown:
      setval      0, result[1]
      ret
```

The EndMovement, AbsMovement, etc. functions (the names are not compulsory) should implement the customized management of the required actions. To ease the programmer's job standard movement functions are provided as a guide to develop customized ones.

Calibration

The customized *calibration* functions must respect the following rules:

- The function must belong to the same subgroup of the referred axis.
- Function name must be **CalibAx*#axis_name*** where axis_name will be changed to the axis name as defined in Configuration. For instance X axis function name will be: CalibAx#X
- The function must provide the following parameters:

1. ***Required action***. May be a point-to-point movement or an interpolated movement.
2. ***Result***. Needed by the system to know whether the required action may be performed by the customized function. If the required action is not supported, the corresponding standard function is used.
3. ***Speed***. Calibration movement speed
4. ***Positive position***. Positive calibration movement position.
5. ***Negative position***. Negative calibration movement position.
6. ***Wait time***. Wait time between subsequent movements.

NOTE: please keep in mind that in some cases actions performed on the calibration control board cause the execution of the axis movement function. For instance at the end of a calibration movement (when the stop button is pressed) an axis status reset is performed calling the customized axis movement function with the "required action" parameter set to _MOVAXEND. The same way when the axis position is modified in the calibration control board the axis movement function is called with the "required action" parameter set to _MOVAXSET.

Custom axis calibration function example:

```
   Function CalibAx#X
      param action as integer
      param result[1] as integer
      param speed as float
      param PosPosition as double
      param NegPosition as double
      param WaitTime as float

      setval      1,result[1]
      select action
      case _CALAXPP
            fcall PPCalibration X, speed, PosPosition, NegPosition, _
                  WaitTime
      case _CALAXINT
            fcall IntCalibration X, speed, PosPosition, NegPosition, _
                  WaitTime
      case else
            call  Unknown
      endselect

      fret

   Unknown:
      setval      0, result[1]
      ret
```

The PPCalibration, IntCalibration etc. functions (the names are not compulsory) should implement the customized management of the required actions. To ease the programmer's job calibration standard functions are provided as a guide to develop customized ones.

Interaction with the window of Manual axis movement

Functions for interaction with the window of manual axis movement should comply with the following specifications:

- The function should be in the same sub-group which belongs to the reference axis
- The function name should be **MoveAx#*axis_name#Action*** where name_axis should be replaced with the axis name defined in the configuration and Action can assume one of the following definitions:

OPEN	indicates that the user has just opened the movement axis window
CLOSE	indicates that the user is going to close the movement axis window
ACTIVE	shows that the movement axis window is active
INACTIVE	shows that the movement axis window is not active
JOG	indicates that a shifting movement managed in runtime by the operator is set
STEP	indicates that a shifting movement with an predefined pitch is set
ABSOLUTE	indicates that a shifting movement with a determined position is set.

For instance, if the axis handling window for X-axes has been opened, the function named MoveAx#X#Open will be called.

Modifying the Window of Manual axis movement

It is possible to add up to 4 buttons to the axis movement window. Some GPL functions with fixed name MoveAx#NomeaAsse#BUTTONtext should be defined in the same sub-group where the concerned axis is defined. NameAxis represents the concerned axis name and test represents the test, that will be displayed on the button.The test can contain the character '&' to introduce a keyboard accelerator. If the test begins with a number between 1 and 4, this number is considered as the position where the button will be inserted in the axis movement window. The button test can be translated, if a DEFMSG with MOVEAX#BUTTONtest as identificator is introduced into the group where the axis is. Pressing the customized button includes the execution of the associated GPL function. Any exiting function delay or any check of function's run start are not executed.

Standard calibration and movement functions

Those shown below are standard functions used by manual movement and calibration control boards.
The functions change depending on axis type: encoder reading, stepper, etc.
The following functions may be customized.

Standard manual movement functions

Absolute position movement

```
; for stepper motor axes
```

```
Function AbsMovement
        param axisname as axis
        param speed as float
        param position as double

        ifstill         axisname goto move
        fret
move:
        setvel          axisname, speed
        movabs          axisname, position
        waitstill       axisname
        fret

; for all other kind of axis
Function AbsMovement
        param axisname as axis
        param speed as float
        param position as double

        iftarget        axisname goto move
        ifstill         axisname goto move
        fret
move:
        setvel          axisname, speed
        movabs          axisname, position
        waitstill       axisname
        fret
```

Incremental movement

```
; for stepper motor axes
Function IncMovement
        param axisname as axis
        param speed as float
        param position as double

        ifstill         axisname goto move
        fret
move:
        setvel          axisname, speed
        movinc          axisname, position
        waitstill       axisname
        fret

; for all other kind of axis
Function IncMovement
        param axisname as axis
        param speed as float
        param position as double

        iftarget        axisname goto move
        ifstill         axisname goto move
        fret
move:
        setvel          axisname, speed
        movinc          axisname, position
        waitstill       axisname
        fret
```

Position setting

```
; for encoder reading axes
Function PositionSet
        param axisname as axis
        param position as double

        setquote        axisname, position
        fret

; for stepper motor axes
Function PositionSet
        param axisname as axis
        param position as double

        ifstill         axisname goto set
        fret
set:
        setquote        axisname, position
        fret

; for all other kind of axis
Function PositionSet
        param axisname as axis
        param position as double

        iftarget        axisname goto set
        ifstill         axisname goto set
        fret
set:
        setquote        axisname, position
        fret
```

Free status setting

```
Function FreeAxis
        param axisname as axis

        free            axisname
        fret
```

Normal status setting

```
Function NormalAxis
        param axisname as axis

        normal          axisname
        fret
```

Calibration standard functions

Point-to-point movements calibration

```
; for stepper motor axes
Function PPCalibration
        param axisname as axis
        param speed as float
        param PosPosition as double
        param NegPosition as double
        param WaitTime as float

        setvel          axisname, speed
loop:
        movabs          axisname, PosPosition
        waitstill       axisname
        delay           WaitTime
        movabs          axisname, NegPosition
        waitstill       axisname
        delay           WaitTime
        goto            loop
        fret

; for all other kind of axis
Function PPCalibration
        param axisname as axis
        param speed as float
        param PosPosition as double
        param NegPosition as double
        param WaitTime as float

        setvel          axisname, speed
loop:
        movabs          axisname, PosPosition
        waitstill       axisname
        ifquotet        axisname,<>,PosPosition goto exit
        delay           WaitTime
        movabs          axisname, NegPosition
        waitstill       axisname
        ifquotet        axisname,<>,NegPosition goto exit
        delay           WaitTime
        goto            loop
exit:
        fret
```

Interpolated movements calibration

```
Function IntCalibration
        param axisname as axis
        param speed as float
        param PosPosition as double
        param NegPosition as double
        param WaitTime as float

        setveli         axisname, speed
loop:
        linearabs       axisname, PosPosition
        waitstill       axisname
        ifquotet        axisname,<>,PosPosition goto exit
        delay           WaitTime
        linearabs       axisname, NegPosition
        waitstill       axisname
        ifquotet        axisname,<>,NegPosition goto exit
        delay           WaitTime
        goto            loop
```

```
exit:
        fret
```

Function OnUIEnd

The function "OnUIEnd#" is performed, if available, by Albatros before ending all the tasks in a module. The function must be defined in the file of intergroup functions. Maximum execution time of the function "OnUIEnd#" is 2 seconds, then Albatros will terminate all the tasks.

Function OnUIPlugged#

The OnUIPlugged# function is executed, when you need to know, for instance, if Albatros, after switching on the plant, is informed of the remote module.
This function must be defined within the intergroup.

Function OnUIUnplugged#

Function "OnUIUnplugged#" is executed before ending the execution of Albatros (and so before Albatros disconnects from a module). This function must be defined within the intergroup. Albatros executes this function within max. 2 seconds.
During this time

- Cycle errors
- System errors
- Messages are read.

At the end of the execution, Albatros closes.

3.2 Instructions

3.2.1 Conventions

The following pages have been organized as files and contain, for each instruction:

- the *Syntax*
- a description of the *arguments*: type of data and admitted values
- a *Description* of functioning
- *Notes*
- *Examples*

All the instructions of the same type have been grouped together, to simplify learning and consultation.

3.2.2 Types of instructions in the GPL language

The language is composed of instructions that can be grouped as follows:

Instructions for Input/Output management

GETFEED	reads the override feed rate
GETVF	reads the voltage/frequency converter
INPANALOG	reads an analog input
INPBCD	reads a series of digital nibbles in BCD format
INPFLAGPORT	reads a flag port
INPPORT	reads a digital port
MULTIINPPORT	reads up to 4 output ports
MULTIOUTPORT	sets up to 4 output ports
MULTISETFLAG	sets several flags on 1
MULTISETOUT	sets several outputs on 1
MULTIRESETFLAG	sets several flags on 0
MULTIRESETOUT	set several outputs on 0
MULTIWAITFLAG	waits for the state of a flag bit or flag switch
MULTIWAITINPUT	waits for the state of various inputs
OUTANALOG	modifies an analog output
OUTBCD	modifies a series of digital nibbles in BCD format

OUTFLAGPORT	modifies a flag port
OUTPORT	modifies a digital port
RESETFLAG	sets a flag on 0
RESETOUT	sets an output on 0
SETFLAG	sets a flag on 1
SETOUT	sets an output on 1
WAITFLAG	waits for the state of a flag bit or flag switch
WAITINPUT	waits for the state of an input
WAITPERSISTINPUT	waits for a persistent state of an input

Instructions for Axes management

CHAIN	chains an axis to another
CIRCABS	absolute circular interpolation
CIRCINC	incremental circular interpolation
CIRCLE	makes a circle
COORDIN	coordinated axis movement
DISABLECORRECTION	disables the linear correction for the specified axis
EMERGENCYSTOP	forces an emergency stop of the axes
ENABLECORRECTION	enables the linear correction for the specified axis
ENDMOV	end of axis movement
FASTREAD	fast axis position read
FREE	sets the axis in free
HELICABS	absolute helicoidal interpolation
HELICINC	incremental helicoidal interpolation
JERKCONTROL	enables or disables interpolation movement control
JERKSMOOTH	links with acceleration and speed continuity, the speed profiles of the axis while contouring.
LINEARABS	absolute linear interpolation
LINEARINC	incremental linear interpolation
MOVABS	absolute movement of axes
MOVINC	incremental movement of axes
MULTIABS	absolute multi-axis linear interpolation
MULTIINC	incremental multi-axis linear interpolation
NORMAL	disables axis free
RESRIFLOC	resets initial reference
SETINDEXINTERP	associates a variable for the counting of executed interpolation
SETLABELINTERP	associates a variable for the identification of a displacement block
SETPFLY	fly homing
SETPFLYCHAINSTRAT	enables control of slave axes behaviour for a master setpfly instruction
SETPZERO	homing on zero
SETPZEROCHAINSTRAT	enables control of slave axes behaviour for a master setpfly instruction
SETQUOTECHAINSTRAT	enables control of slave axis behaviour for a setquote instruction on the master
SETRIFLOC	set spacial reference points
SETTOLERANCE	sets the tolerance values for the linear interpolation
START	restarts axis movement
STARTINTERP	forces start of an interpolation
STOP	interrupts axis movement
SWITCHENC	allows replacing the encoder of an axis with that of another axis
SYNCROOPEN	opens a synchronized movement channel
SYNCROCLOSE	closes the synchronized movement channel
SYNCROMOVE	assigns a synchronized movement point
SYNCROSETACC	sets the acceleration for synchronized movements
SYNCROSETDEC	sets the acceleration for synchronized movements
SYNCROSETVEL	sets the acceleration for synchronized movements
SYNCROSETFEED	sets the axes speed for a synchronized movement
SYNCROSTARTMOVE	starts processing a synchronized movement

Instructions for the management of Timers and Counters

DECOUNTER	decrements a counter
HOLDTIMER	locks a timer
INCOUNTER	increments a counter
SETCOUNTER	sets a counter
SETTIMER	sets a timer
STARTTIMER	starts the timer

Instructions for Communications' management

CLEARRECEIVE	empties the list of RECEIVE to satisfy
COMCLEARRXBUFFER	empties inbox buffer of a serial port
COMCLOSE	closes a serial port
COMGETERROR	reads the error code
COMGETRXCOUNT	reads the number of bytes in inbox buffer
COMOPEN	opens a serial port
COMREAD	reads from the serial port
COMREADSTRING	reads a string from the serial port
COMWRITE	writs on the serial port
COMWRITESTRING	writes a string on the serial port
RECEIVE	external data reception
SEND	sends data from outside
WAITRECEIVE	external data reception with standby

Instructions for Mathematical management

ABS	absolute value
ADD	sum
AND	AND binary
ARCCOS	arc cosine
ARCSIN	arc cosine
ARCTAN	arc tangent
COS	cosine
DIV	division
EXP	exponential
EXPR	resolves mathematical expressions
LOG	natural logarithm
LOGDEC	base 10 logarithm
MOD	module
MUL	multiplication
NOT	binary NOT
OR	binary OR
RANDOM	generates a random number
RESETBIT	sets a bit on 0
ROUND	rounds
SETBIT	sets a bit on 0
SHIFTL	rotates the bits to left
SHIFTR	rotates the bits to right
SIN	sine
SQR	square root
SUB	subtraction
TAN	tangent
TRUNC	truncation
TYPEOF	type of the argument
XOR	binary XOR

Instructions for Multitask management

ENDMAIL	reports the end of the execution of a task
ENDREALTIMETASK	terminates a realtime task
ENDTASK	terminates a task
GETPRIORITYLEVEL	reads the priority level of the current task
GETREALTIME	returns time lapsed since the beginning of axis realtime
GETREALTIMECOUNT	returns the number of RealTime lapsed
HOLDTASK	interrupts the execution of a task
RESUMETASK	resumes the execution of a task
SENDIPC	sends an IPC information
SENDMAIL	sends a command to the 'mail' mailbox
SETPRIORITYLEVEL	sets the priority level of the current task
STARTREALTIMETASK	starts a realtime task
STARTTASK	starts the execution of a task
STOPTASK	stops the execution of a task and interrupts the movement of the associated axes
WAITIPC	waits for an IPC information
WAITMAIL	receives a command from the 'mail' mailbox
WAITTASK	waits for a task to terminate

Instructions for Matrix management

CLEAR	sets variable, vector and matrix to zero
FIND	searches for an element
FINDB	searches for an element in a vector or in a matrix increasingly ranged
LASTELEM	last element of a vector or of a matrix
LOCAL	declaration of a local variable, vector, local matrix
MOVEMAT	copies the row of a matrix in another
PARAM	declaration of a function parameter
SETVAL	changes a variable
SORT	sorts vector or matrix

Instructions for Flux management

CALL	calls a subprogram
DELONFLAG	disables the emergency management on flag bit or flag switch
DELONINPUT	disables the emergency management on digital input
ENDREP	end of the block repetition with REPEAT
FCALL	calls a function
FOR	extension of REPEAT
FRET	return from call to function
GOTO	jumps to a label
IF	test on a variable
IFACC	tests, if the axis is accelerating
IFAND	test on AND operation
IFBIT	test on bits
IFBLACKBOX	tests if the record of the logical device activity is active.
IFCHANGEVEL	tests, if the axis is changing speed
IFCOUNTER	test on a counter
IFDEC	tests if the axis is decelerating
IFDIR	test on axis direction
IFERRAN	test on loop error
IFERROR	test on active cycle error
IFFLAG	test on a flag
IFINPUT	test on an input
IFMESSAGE	test on the active message
IFOR	test on OR operation
IFOUTPUT	test on an output
IFQUOTER	test on real position
IFQUOTET	test on real position
IFRECEIVED	test on data reception
IFREG	tests if the axis is in steady-state conditions
IFSAME	verifies that both arguments refer to the same data
IFSTILL	tests if the axis is still
IFSTR	test on a string

IFTARGET	tests if the axis has reached the target
IFTASKHOLD	tests if the parallel function is interrupted
IFTASKRUN	tests if the parallel function is running
IFTIMER	test on a timer
IFVALUE	test on a variable
IFVEL	test on axis speed
IFWIN	tests if the axis is in the window
IFXOR	test on XOR operation
NEXT	end of block repetition with FOR
ONERRSYS	sets the call to a function on a system error
ONFLAG	emergency in flag bit or flag switch
ONINPUT	emergency in digital input
REPEAT	repetition of an instruction block
RET	return from subprogram
SELECT	multiple selection with jump
TESTIPC	verifies the presence of an IPC information
TESTMAIL	test and reception of a command
WAITACC	waits for axis acceleration
WAITCOLL	waits for the axis to exceed a position from which it should start checking the presence of a collision
WAITDEC	waits for axis deceleration
WAITREG	waits for the axis to be in steady-state conditions
WAITSTILL	waits for the final position to equal the target position
WAITTARGET	waits for the axis to reach the target
WAITWIN	waits for the axis to be in the window

Instructions for String management

ADDSTRING	chains two strings
CONTROLCHAR	sets a control character in a string variable
LEFT	extracts the first characters
LEN	reads the length of a string
MID	extracts some characters
RIGHT	extracts the last characters
SEARCH	searches for a string
SETSTRING	modifies a string variable
STR	converts from number to a string
VAL	converts from string to number

Instructions for axis Parameter management

DYNLIMIT	enables or disables dynamically the test on axis limit exceeding.
ENABLESTARTCONTROL	enables and sets the timeout to control the non-start up or the sudden stop of the axis
DEVICEID	writes the logical address associated to a device
GETAXIS	reads one or more data of an axis
LOOKAHEAD	sets the interpolation lookahead
NOTCHFILTER	sets the notch filter cut-off frequency for the specified axis
RATIO	sets the chaining ratio of a slave axis with respect to its own master
RESLIMNEG	disables the negative limit of the axis
RESLIMPOS	disables the positive limit of the axis
SETACC	sets acceleration
SETACCI	sets the acceleration for interpolation
SETACCLIMIT	enables and disables the automatic calculation of the interpolation steady state speed
SETACCSTRATEGY	sets the type of acceleration
SETADJUST	sets the adjust of an axis
SETAXPARTYPE	changes the axis parameter set currently in use
SETBACKLASH	decreases or deletes the effects of the mechanical blacklash on the axis trajectory
SETBIGWINFACTOR	modifies the multiplication factor for the calculation of the big window on the requested axis.

SETCONTORNATURE	sets the contouring angle
SETDEC	sets the deceleration
SETDECI	sets the deceleration for the interpolation
SETDEADBAND	sets the minimum voltage for the affected axis
SETDERIV	sets the coefficient of derived action
SETDERIVI	sets the coefficient of interpolation derived action
SETDYNRATIO	changes dynamically the chaining ratio during the movement of the master axis.
SETFEED	sets the point-to-point feed rate
SETFEEDCOORD	sets the percentage value of the highest instantaneous variation of the axis feed rate.
SETFEEDF	sets the feed forward
SETFEEDFA	sets the acceleration feed forward
SETFEEDFAI	sets acceleration feed forward in interpolation
SETFEEDI	sets feed forward in interpolation
SETFEEDFI	sets the feed forward in interpolation
SETINDEXEN	enables or disables on the axis the reset of the position that corresponds to the zero position reference
SETINTEG	sets the coefficient of integral action
SETINTEGI	sets coefficient of interpolation integral action
SETINTEGTIME	it sets the number of loop error samples used to calculate the integral component
SETIRMPP	sets the speed of start ramp
SETLIMNEG	sets the negative limit of the axis
SETLIMPOS	disables the positive limit of the axis
SETMAXER	sets the highest tolerated tracking value
SETMAXERNEG	sets highest tolerated tracking value (negative direction)
SETMAXERPOS	sets highest tolerated tracking value (positive direction)
SETMULTIFEED	sets the percentage value of feed rate override of the affected axes .
SETOFFSET	enables a position offset
SETPHASESINV	enables or disables on the affected axis thye phase inversion
SETPROP	sets the coefficient of proportional action
SETPROPI	sets coefficient of interpolation proportional action
SETQUOTE	sets the position
SETREFINV	enables or disables on the affected axis the inversion of speed reference
SETRESOLUTION	changes the resolution if an axis
SETSLOWPARAM	changes the parameters to calculate the slowdown speed in the event that the slowdown functionality while contouring is active
SETVEL	sets the speed
SETVELI	sets the interpolation speed
SETVELILIMIT	sets the individual components of the affected axis speed

Various instructions

CLEARERRORS	deletes all the module cycle errors
CLEARMESSAGES	deletes all messages of the module
DEFMSG	defines a group message
DELAY	locks the current function for a period of time
DELERROR	deletes ma previous cycle error
DELMESSAGE	deletes a previous message
ERROR	sends a cycle error to the PC
IFDEF/ELSEDEF/ENDDEF	test for the conditional compilation
MESSAGE	sends a message to the PC
SYSFAULT	disables SYSOK signal
SYSOK	enables SYSOK signal
WATCHDOG	enables, updates, disables the whatchdog from GPL on the TMSWD hardware module

Instructions for SLM management

SLMCOMMAND	executes a SLM command
SLMEEPROMDISABLE	execute an EEPROM writing disabling command
SLMEEPROMENABLE	executes an EEPROM wiring enabling command
SLMGETEEPROM	reads an EEPROM memory location
SLMGETPARAM	reads a SLM parameter
SLMGETREGISTER	reads a SLM register
SLMGETSTATUS	reads a drive quantity
SLMSETEEPROM	writes an EEPROM memory location
SLMSETPARAM	sets a SIM parameter
SLMSETREGISTER	sets a SLM register

Instructions for CANopen management

CANOPENDRIVER	opens a CANopen communication canal
CANCLOSEDRIVER	closes a CANopen communication channel
CANRESETBOARD	resets a CANopen board
CANSETOBJECT	writes a CANopen object
CANGETOBJECT	reads a CANopen object
GETCNSTATE	returns the NMT protocol state for the node of a CANOpen board.
GETSDOERROR	returns the last error occurred
GETMNSTATE	returns the status of the NMT protocol for the master node of the CANOPen board.
SENDPDO	writes the content of an asynchronous PDO
SETNMTSTATE	sets the status of the NMT protocol for the node of the CANOpen board.
RECEIVEPDO	reads the content of an asynchronous PDO

Instructions for Mechatrolink II management

MECCOMMAND	sends a command to the axis drive
MECGETPARAM	reads a parameter of the indicated axis
MECSETPARAM	writes a parameter in the indicated axis
MECGETSTATUS	reads the values of STATUS, ALARAM and IO_MON

Instructions for Simulation

ENABLE	enables one or more axes
ENABLEFORCEDINPUT	enables the inputs to be forced
DISABLE	disables one or more axes
DISABLEFORCEDINPUT	disables the inputs to be forced
SETFORCEDINPUT	forces an input to ON
RESETFORCEDINPUT	forces an input to OFF
SETFORCEDBCD	forces a nibble set in BCD format
SETFORCEDPORT	forces an input port
SETFORCEDANALOG	forces an analog input

Instruction for the "Blackbox" functionalities

ENDBLACKBOX	ends the record functionality
PAUSEBLACKBOX	interrupts the record functionality
STARTBLACKBOX	starts the record functionality

Instructions for Powerlink II and EtherCAT management

AXCONTROL	sets a value for ControlWord

ACTIVATEMODE	sets an operating mode
AXSTATUS	returns the value in the StatusWord
CNBYDEVICE	returns the EPL coordinates of a device
GETPDO	returns an object inside a PDO Ethercat
HOMING	searches the "zero position"
READDICTIONARY	reads the content of a dictionary object
SETPDO	sets an object inside a PDO Ethercat
WRITEDICTIONARY	writes the content of the dictionary object

Instructions for ISO control

ISOG0	sets the rapid movement
ISOG1	sets the interpolated movement
ISOG9	sets the forced stop of the movement
ISOG90	sets the interpretation of the positions as absolute positions
ISOG91	sets the interpretation of the positions as relative positions
ISOG93	sets the interpretation as inverse of the time
ISOG94	sets the interpretation of the speed as unit of measure per minute
ISOG216	defines the matrices for machine parametrisation
ISOG217	describes the physical axes and the virtual axes, which make up the machine
ISOM2	frees the axes free from ISO movement
ISOM6	selects the indices of parametrisation matrices
ISOSETPARAM	sets some parameters that characterize the fluidity of the ISO interpolation movement.
KINEMATICEXPR	sets the single expressions of inverse and direct kinematics

3.2.3 Input/Output

GETFEED

Syntax

GETFEED **variable**

Arguments

variable feed rate

Description

It copies the value of the feed rate read from the remote I/O card, in the specified **variable**.
Feedrate value is included between 0 and 100 and it is a percentage value.
It operates on an analog input which is not visible in configuration.
On a Albnt board this is the connector of the 4th axis (red) which, when the card has been appropriately configured, acts as an analog input (grey/greywhite wires).
For Cn2004 board the feed rate is managed by configuring the first analogical input AIN1. For all the other T.P.A boards controlling the feed rate a dedicated connector is available.

GETVF

Syntax

GETVF **variable**

Arguments

variable integer variable

Description

It reads the voltage/frequency converter value normally used to manage the feed rate override and puts the result in the specified **variable**. The read value interval ranges from 0 to 16000, which corresponds to an input voltage of 0 - 8 volts. For example the value 8000 corresponds to 4 volts.

INPANALOG

Syntax

INPANALOG	**inpanalogname, variable**

Arguments

inpanalogname	name of analog input device
variable	variable

Description

It copies the value of the analog input specified by **inpanalogname** in the specified **variable**.

INPBCD

Syntax

INPBCD	**digitname1 [,digitname2, …], variable**

Arguments

digitname1	name of nibble device
variable	variable

Description

It reads the input nibbles specified by the **digitname** arguments (from **1** to **4 max**). It reads each nibble as a number, where argument **digitname1** has the highest weight, and it sets the value of the number in the **variable**.
In practice it is used to read decimal numbers from physical devices which indicate them as groups of 4 inputs (nibble). The inputs of each nibble correspond to the bits necessary to represent the decimal number in the binary system.

INPFLAGPORT

Syntax

INPFLAGPORT	**flagportname, variable**

Arguments

flagportname	name of flag port device
variable	variable

Description

It copies the state of the flag port specified by **flagportname** in the specified **variable.**
The flag port is detected as a bit mask. A bit is associated to each flag of the port. If a flag is "ON", the corresponding bit is set on 1.

INPPORT

Syntax

INPPORT	**portname, variable**

Arguments

portname	name of input port device
variable	integer or char variable

Description

It copies the state of the **portname** input port in the specified **variable**.
The input port is detected as a bit mask. If the input of the port is "ON" the corresponding bit is set on 1.

MULTIINPPORT

Syntax

MULTIINPPORT	**port1[,…,port4],variable**

Arguments

port1	provides the bits from 0 to 7
port2	provides the bits from 8 to 15
port3	provides the bits from 16 to 23
port4	provides the bits from 24 to 31
variable	integer variable receiving the input ports

Description

It reads no more than 4 output ports at the same time and writes them into a **variable**. Ports are read atomically. This procedure guarantees that the ports are read within the same real-time. Port1 corresponds to the lower byte, port4 corresponds to the greater byte.

Port4	Port3	Port2	Port1

31 23 15 7 0 bit

MULTIOUTPORT

Syntax

MULTIOUTPORT **value, portname1[,...,portname4]**

Arguments

value	number or integer value to be written in the output ports
portname1	receives the bits from 0 to 7
portname2	receives the bits from 8 to 15
portname3	receives the bits from 16 to 23
portname4	receives the bits from 24 to 31

Description

It writes the **value** into four output ports at the same time. Ports are read atomically. This procedure guarantees that the ports are written within the same real-time. If **portname2**, **portname3**, **portname4** are not specified, the value of the byte is 0.

Port4	Port3	Port2	Port1

31 23 15 7 0 bit

MULTIRESETFLAG

Syntax

MULTIRESETFLAG **mask, flagname1[, ..., flagname32]**

Arguments

mask	mask of involved flags - constant or variable
flagname1	name of flag device

Description

It disables, that is, it switches to "OFF", all the **flagnames** (1÷32), whose bit is set on 1 in the argument **mask**.
The **mask** 0 bit (lowest weight) corresponds to **flagname1**.

MULTIRESETOUT

Syntax

MULTIRESETOUT **mask, outputname1[, ..., outputname32]**

Arguments

mask	mask of involved outputs - constant or variable
outputname1	name of output device

Description

It disables all the **outputnames** (1÷32), whose bit in the argument **mask** is set on 1. The **mask** 0 bit (lowest weight) corresponds to **outputname1**.

MULTISETFLAG

Syntax

MULTISETFLAG **mask, flagname1[, ..., flagname32]**

Arguments

mask	mask of involved flags - constant or variable
flagname1	name of flag device

Description

It enables, that is, it switches to "ON", all the **flagnames** (1÷32), whose bit in the argument **mask** is set on 1. The **mask** 0 bit (lowest weight) corresponds to **flagname1**.

MULTISETOUT

Syntax

MULTISETOUT **mask, outputname1[, ..., outputname32]**

Arguments

mask	mask of involved outputs - constant or variable
outputname1	name of output device

Description

It enables all the **outputname** outputs (1÷32), whose bit in the argument **mask** is set on 1.
The 0 bit of **mask** (lowest weight) corresponds to **outputname1**. If the output is a monostable output it is disabled automatically after 200 milliseconds.

MULTIWAITFLAG

Syntax

MULTIWAITFLAG **mask, flag1[, ..., flag32], state [, timeout [, GOTO label]]**
MULTIWAITFLAG **mask, flag1[, ..., flag32], state [, timeout [, CALL subprogramname]]**
MULTIWAITFLAG **mask, flag1[, ..., flag32], state [, timeout [, functionname]]**

Arguments

mask	constant or variable. Mask of involved flags
flag1[,...flag32]	name of flag device
state	predefined constant. Acceptable values are: **ON** flag state: enabled **OFF** flag state: disabled
timeout	constant or variable. Maximum wait time.
label	jump to label (GOTO)
subprogramname	subprogram label (CALL)
functionname	name of function

Description

It waits for the specified flags, from **flag1...flag32** to be in the state indicated by the **state** parameter (ON/OFF).
It checks all the flags whose bit in the argument **mask** is enabled (ON). The 0 bit of the argument **mask** (lowest weight) corresponds to the bit defined by **flag1**, the 1 bit corresponds to the bit defined by **flag2** and so on, up to the bit defined by **flag32**.
The **timeout** parameter allows you to set a different timeout from default timeout which waits one second.
When **label**, **subprogramname** or **functionname** are present, at the end of timeout the program jumps to **label** or calls **subprogramname** or **functionname**.

MULTIWAITINPUT

Syntax

MULTIWAITINPUT	**mask, input1[, ..., input32], state [, timeout [, GOTO label]]**
MULTIWAITINPUT	**mask, input1[, ..., input32], state [, timeout [, CALL subprogramname]]**
MULTIWAITINPUT	**mask, input1[, ..., input32], state [, timeout [, functionname]]**

Arguments

mask	constant or variable. Mask of involved inputs
flag1[,...flag32]	name of input
state	predefined constant. Acceptable values are: **ON** flag state: enabled **OFF** flag state: disabled
timeout	constant or variable. Maximum wait time.
label	jump to label (GOTO)
subprogramname	subprogram label (CALL)
functionname	name of function

Description

It waits for the specified inputs, from **input1...input 32** to be in the state indicated by the **state** parameter (ON/OFF).
It verifies all the inputs whose bit in the argument **mask** is enabled (ON). The 0 bit of the argument **mask** (lowest weight) corresponds to the bit defined by **input1**, the 1 bit corresponds to the bit defined by **input2** and so on, up to the bit defined by **flag32**.
If no optional arguments are specified, a second after the beginning of instruction execution (default time), the following parametrised message appears: "Wait inputn ON/OFF". The name of the indicated input corresponds to the first enabled input which still has not satisfied the state. If the **timeout** parameter is included, the above mentioned message will appear when the set timeout expires. If the requested condition takes place, when timeout has expired, a parametrised message will appear automatically to delete the previous one.
When **label**, **subprogramname** or **functionname** are present, at the end of timeout the program jumps to **label** or calls **subprogramname** or **functionname**

OUTANALOG

Syntax

OUTANALOG	**outanalogname, value**

Arguments

outanalogname	name of analog output device or axis
value	constant or variable

Description

It sets the analog output or the axis indicated by **outanalogname** to the voltage specified by **value.**

OUTBCD

Syntax

OUTBCD	**digitname1 [,digitname2, ...], variable**

Arguments

digitname	name of nibble device
variable	constant or variable

Description

In computing and electronic systems, binary-coded decimal (BCD) is a class of binary encodings of decimal numbers, where each decimal digit is represented by a binary code of four bits, whose value ranges from 0 (0000) to 9 (1001).

This instruction converts the decimal value contained in the **variable** to a sequence of numbers. Each digit is converted to the binary system and the bit mask thus obtained is set in the corresponding nibble. The digit with the highest weight is associated to the first nibble **(digitname1)**.

Example

```
OUTBCD          nib1,nib2,nib3 234

 ; 4 in binary is 0100 and lights the third led of the nibble3
 ; 3 in binary is 0011 and lights the first and the second led of the
 nibble2
 ; 2 in binary is 0010 and lights the second led of the nibble1
```

OUTFLAGPORT

Syntax

OUTFLAGPORT **flagportname, value**

Arguments

flagportname	name of the flag port device
value	constant or variable

Description

It copies the **value** in the flag port specified by **flagportname**.
The **value** parameter is detected as a bit mask. Each bit is associated to a port flag. If the bit is set on 1 the flag is "ON".

OUTPORT

Syntax

OUTPORT **portname, value**

Arguments

portname	name of output port device
value	constant or variable, integer or char

Description

It copies the **value** in the **portname** output port.
The ouptput port is detected as a bit mask. It the bit is set on 1 the corresponding output is on "ON".

RESETFLAG

Syntax

RESETFLAG **flagname**

Arguments

flagname	name of flag device

Description

It disables (switches to OFF) the **flagname** flag.

RESETOUT

Syntax

RESETOUT **nameoutput**

Arguments

nameoutput	name of digital output device

Description

It disables (switches to OFF) the **nameoutput** output.

SETFLAG

Syntax

SETFLAG **flagname**

Arguments

flagname	name of flag device

Description

It enables (switches to ON) the **flagname** flag.

SETOUT

Syntax

SETOUT **nameoutput**

Arguments

nameoutput	name of digital output device

Description

It enables (switches to ON) the **nameoutput** output.
If the output is configured as monostable it is automatically disabled after a 200 millisecond timeout.

WAITFLAG

Syntax

WAITFLAG **flagname, state [, timeout [, GOTO label]]**
WAITFLAG **flagname, state [, timeout [, CALL subprogramname]]**
WAITFLAG **flagname, state [, timeout [, functionname]]**

Arguments

flagname	name of flag device
state	predefined constant. Acceptable values are: **ON** flag state: enabled **OFF** flag state: disabled
timeout	constant or variable. Maximum wait time.
label	jump label (GOTO)
subprogramname	subprogram label (CALL)
functionname	name of function

Description

It waits for the flag **flagname** to be in the state indicated by the parameter **state** (ON/OFF).
If the only optional argument present is **timeout**, the cycle error "**flagname** flag awaiting **state**" is generated at end of timeout.
If the condition is satisfied after timeout expiry, the cycle error previously sent out for that task is automatically cancelled.
When **label**, **subprogramname** or **functionname** are present, at the end of timeout the program jumps to **label** or calls **subprogramname** or **functionname** without generating any automatic display.

Note

To avoid waiting for flags during work cycles, we suggest setting a timeout.

WAITINPUT

Syntax

WAITINPUT **nameinput, state [, timeout [, GOTO label]]**
WAITINPUT **nameinput, state [, timeout [, CALL subprogramname]]**
WAITINPUT **nameinput, state [, timeout [, functionname]]**

Arguments

nameinput	name of input
state	predefined constant. Acceptable values are: **ON** flag state: enabled **OFF** flag state: disabled
timeout	constant or variable. Maximum wait time.
label	jump label (GOTO)
subprogramname	subprogram label (CALL)
functionname	name of function

Description

It waits for the **nameinput** input to be in the state indicated by the parameter **state** (ON/OFF).
If no optional arguments are specified, the cycle error "**Nameinput** digital input awaiting **state**" is generated automatically 20 seconds after the beginning of instruction execution. If the only optional argument present is **timeout**, the above mentioned message is generated at the end of timeout.
If the condition is satisfied after **timeout** expiry, the cycle error previously sent out for that task is automatically cancelled.
If **label**, **subprogramname** or **functionname** are present, when timeout expires the program jumps to **label** or calls **subprogramname** or **functionname** without generating any automatic display.

Note

To avoid having to wait for input signals during a work cycles, we suggest setting a shorter timeout than default time (20 seconds).

Example

Routine of Axis Homing

WAITPERSISTINPUT

Syntax

WAITPERSISTINPUT	**nameinput, state, timepersist [, timeout [, GOTO label]]**
WAITPERSISTINPUT	**nameinput, state, timepersist [, timeout [, CALL subprogramname]]**
WAITPERSISTINPUT	**nameinput, state, timepersist [, timeout [, functionname]]**

Arguments

nameinput	name of digital input device
state	predefined constant. Acceptable values are: **ON** flag state: enabled **OFF** flag state: disabled
timepersist	constant or variable
timeout	constant or variable. Maximum wait time.
label	jump label (GOTO)
subprogramname	subprogram label (CALL)
functionname	name of function

Description

It waits for the **nameinput** input to reach the state indicated by the parameter **state** (ON/OFF) and to remain in that state for the time specified in **timepersist** (unit of measure: seconds).
If no optional arguments are specified, the cycle error "**Nameinput** digital input awaiting **state**" is generated automatically 20 seconds after the beginning of instruction execution.
If the only optional argument present is **timeout**, the above mentioned message is generated at the end of timeout.
If the condition is satisfied after **timeout** expiry, the cycle error previously sent out for that task is automatically cancelled.
When **label**, **subprogramname** or **functionname** are present, at the end of timeout the program jumps to **label** or calls **subprogramname** or **functionname** without generating any automatic display.

Note

To avoid having to wait for input signals during work cycles, we suggest setting a shorter timeout than default time (20 seconds).

3.2.4 Axes

CHAIN

Syntax

CHAIN	**master_axis, slave_axis1 [, ...slave_axis5]**

Arguments

master_axis	name of axis device functioning as master
slave_axis1...slave_axis5	name of axis device functioning as slave

Description

After executing this instruction, the **slave_axes** (1÷5) will execute will execute movements linked to those of the master axis by the chaining ratio set with the RATIO instruction. Both point-to-point and interpolated movements will be chained.
Slave_axis1 is not an optional parameter, it must always be defined.
If a slave axis is to be chained, it can not be engaged in an interpolation and can not be master of other slaves.
In his turn, the master axis cannot be the slave of other axes.
Chaining can be carried out both with positioned axes and moving axes.
To disable axes chaining it is sufficient to execute the instruction NORMAL on the master axis. This last operation can be carried out both with axes in position and with axes in motion. When the chain is disabled while the axes are in motion, the slave gradually decelerates and stops.
A maximum of 8 master axes can be simultaneously defined.
The instruction can be performed also with step-by-step axes (stepper), as long as they can be controlled through TRS_AX.
In addition, all the axes must have a real and not simulated encoder, otherwise the system error no. "4101 - Inconsistent axis AxisName management" is generated".
See also RATIO.

Example

```
CHAIN           X, Y        ; Y axis is chained to X
MOVINC          X, 100      ; X axis moves. Y axis replicate
                            ; X axis movement
```

CIRCABS

Syntax

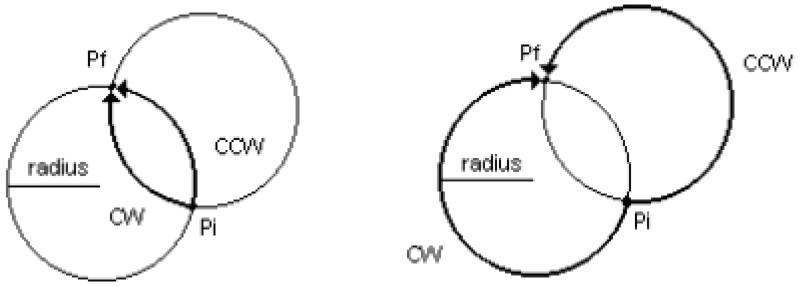
CIRCABS **[label],axis1, position1, axis2, position2, direction, ±radius [, angle]**

Arguments

label	constant or variable integer. Label identifying a displacement bloc
axis1, axis2	name of axis devices
positon1, position2	constant or variable. It indicates the absolute move position
direction	integer variable. It specifies the kind of rotation. Acceptable values are: **CW** clockwise **CCW** anti clockwise
radius	constant or variable. It indicates the value of the radius of the circle.
angle	constant or variable. It indicates the angle of the starting point

Description

2 axes circular interpolation with *absolute transfer* based on programmed positions: position1, position2.
The arch is determined by the starting point (current point), the final point, the value of the **radius** and the **direction**.
The sign applied to the **radius** allows you to select the minor arch (+radius) or the major arch (-radius).
In the rare case in which the starting position of axis1 coincides with **position1** final position and the starting position of axis2 coincides with the **position2** final position a complete circle is drawn. In this case it is necessary ti indicate the argument **angle**, having the same meaning as the instruction CIRCLE (to be referred to).
The angle parameter is necessary to determine precisely the centre of the circle, with the same meaning as the instruction CIRCLE. It is only used when, before instruction execution, **position1** and **position2** coincide with the current position of the axes.
The optional parameter **label** is used in association with the instruction SETLABELINTERP to indentify univocally the displacement bloc
Step-by-step axes can only be used in this instruction, if they are controlled by a TRS-AX remote.
In this case it must be taken into account that the word interpolation refers to a coordinated movement of more axes affected by discrete error due to axis piloting method.



+ radius

- radius

CIRCINC

Syntax

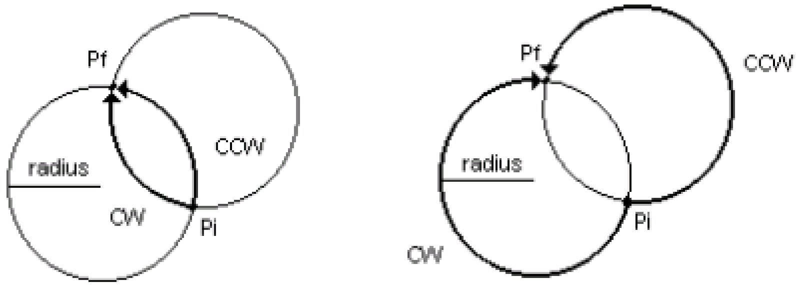
CIRCINC **[label],axis1, position1, axis2, position2, direction, ±radius [, angle]**

Arguments

label	constant or variable integer. Label identifying a displacement bloc
axis1, axis2	name of axis devices
positon1, position2	constant or variable. It indicates the incremental move position
direction	integer variable. It specifies the kind of rotation. Acceptable values are: **CW** clockwise **CCW** anti clockwise
radius	constant or variable. It indicates the value of the radius of the circle
angle	constant or variable. It indicates the angle of the starting point

Description

2 axes circular interpolation with *incremental transfer* based on programmed positions **position1** and **position2**.
The arch is determined by the starting point (current point), the final point, the value of the **radius** and the **direction**.
The sign applied to the **radius** allows you to select the minor arch (+radius) or the major arch (-radius).
In the rare case in which position1 = position2 = 0, a complete circle is drawn. In this case it is necessary to indicate the argument **angle**, with the same meaning as the instruction CIRCLE (to be referred to).
The angle parameter is necessary to determine precisely the centre of the circle, with the same meaning as the instruction CIRCLE. The optional parameter **label** is used in association with the instruction SETLABELINTERP to indentify univocally the displacement bloc.
Step-by-step axes can only be used in this instruction, if they are controlled by a TRS-AX remote. In this case it must be taken into account that the word interpolation refers to a coordinated movement of more axes affected by discrete error due to axis piloting method.



+ radius

- radius

CIRCLE

Syntax

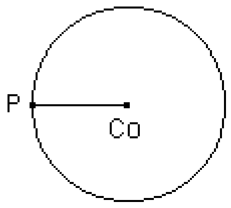
CIRCLE **[label],axis1, axis2, direction, radius, angle**

Arguments

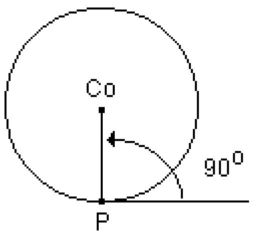
label constant or variable integer. Label identifying a displacement bloc
axis1, axis2 name of axis devices
direction integer variable. It specifies the kind of rotation. Acceptable values are:
CW clockwise
CCW anti clockwise
radius constant or variable. It indicates the value of the radius of the circle.
angle constant or variable. It indicates the angle of the starting point.

Description

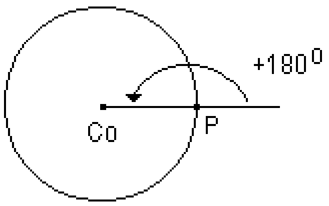
Complete circular interpolation.
It generates a circle with **axis1** and **axis2**, in the indicated direction, with the indicated **radius** and according to the set starting **angle**.
The **radius** can only have positive values.
The **angle** must be given according to the trigonometric convention, positive, clockwise, starting from the X axis. The position of the centre $C_0$of the circle is determined by specifying the angle formed by the radius passing from the programmed initial point P (current point) and the horizontal direction X+. The optional parameter **label** is used in association with the instruction SETLABELINTERP to indentify univocally the displacement bloc.
Step-by-step axes can only be used in this instruction, if they are controlled by a TRS-AX remote. In this case it must be taken into account that the word interpolation refers to a coordinated movement of more axes affected by discrete error due to axis piloting method.



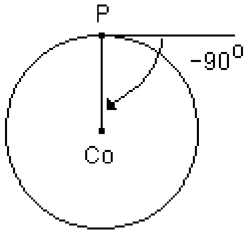
angle = 0



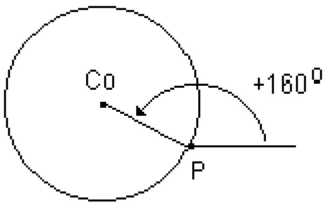
angle = 90⁰



angle = +180⁰



angle = -90⁰ (o+ 270⁰)



angle = +160⁰

COORDIN

Syntax

COORDIN **matrix, value deltaT, direction, begin, end, mask, axis1, n°_column_axis1**
[, (axis2, n°_column_axis2) ÷ (axis32, n°_column_axis32)]

Arguments

matrix data matrix
value deltaT constant or variable. Time basis
direction predefined constant. Direction of data reading in matrix
UP from the last row, upwards
DOWN from the first row downwards
begin global integer variable. The number of the first row
end global integer variable. The number of the last row

mask	axis mask to be enabled
axis1 [....axis32]	name of axis devices
n° _column_axis1[...n_column_axis32]	number of matrix column referring to axis

Description

This instruction allows you to carry out synchronised movements of axes **axis1**, **axis2**, etc. by means of incremental transfers (microvectors) defined by a data **matrix**.
The parameters **axis1** and **n_column_axis1** must always be defined.
The values contained in the **matrix** indicate the absolute positions reached by the various axes one at a time.
Relative incremental transfers (interval between the position of row (n) and row (n-1)) are executed in a lapse of time equivalent to a **multiple** of the time basis (1 ms = Real Time of axes refresh) specified by the argument **value** Δ**t,** which must consequently be expressed by an integer number. When the value of this time has been defined, the distance covered at each movement by each axis determines its speed. This instruction allows you to coordinate the movement of a maximum of 32 axes, along any curved line in space, as generated by SPLINE techniques.
It is not necessary to wait until the instruction is completed; it does not need the STARTINTERP instruction to start. However, a WAITSTILL instruction should be brought to its end, in order to wait for the correct arrival phase of the axes. Possible changes of the feedrate override should be made by means of the SETFEEDI instruction and worked through the SETFEEDCOORD istruction.
The parameter **direction** allows you to determine the direction of the matrix, allowing you to execute the trajectory in both directions.
The columns of the matrix to be scanned can be float or double but not both at the same time.

In addition to the movement of axes along a finite path (defined by the number of matrix rows), infinite movement can be selected using:

- one matrix of a single row. With this operating mode, the control always reads the only row of the matrix and applies the coordinates in the row to the axes. To move the axes, the matrix row should be changed, preferably using a real-time task which guarantees coordinates updating is synchronised with the axes refresh frequency. With this operating mode, the control always reads the only row of the matrix and applies the coordinates in the row to the axes. To move the axes, the matrix row should be changed, preferably using a real-time task which guarantees coordinates updating is synchronised with the axes refresh frequency;
- a matrix of more rows. It is possible to scan the matrix with cycles from the first to the last row indefinitely by setting the values **ini** = 1, **fin** = 0 and **direction** = UP. If a single multi-row matrix row must be executed, it is necessary to set parameters **ini**, **fin** and **direction** in the following way: **ini** = numer of row that must be executed, **fin** = number of row preceding row that must be executed, **direction** = UP. In other case a system error is generated.
  Step-by-step axes can only be used in this instruction, if they are controlled by a TRS-AX remote.

DISABLECORRECTION

Syntax

DISABLECORRECTION **axis [, axis1, ..., axis6]**

Arguments

axis	name of axis device
axis1, ..., axis6	name of axis device

Description

Disables the linear correction for the specified **axis**.
The first parameter is the axis whose correction is to be deactivated, if it is the only parameter specified all the corrections present in the configuration are deactivated. The following parameters allow the specification of which corrections are to be deactivated, if one of these coincides with the first parameter the auto-correction is deactivated.
For a more detailed description see ENABLECORRECTION.

Example

```
; disables only the auto-correction for axis X
DISABLECORRECTION     X, X

; disables the crossed correction (towards X and Y) for axis Z,
; but not the auto-correction
DISABLECORRECTION     Z, X, Y
```

EMERGENCYSTOP

Syntax

EMERGENCYSTOP	**axis, time**

Arguments

axis	name of axis devices
time	constant or integer variable. Ramp time (ms)

Description

It stops the specified axis and any axes possibly involved with it in the interpolated movement. The movement is stopped by a deceleration ramp over the time indicated by the variable **[time].**
In the point-to-point movements if the time set is greater than the configured deceleration time, this latter is used.
In the interpolated movements, if the time set is greater than the maximum value of the deceleration times of all axes involved, the maximum time configured is used.
The movement can be resumed by a START instruction.
The instruction cannot be used if **[axis]** is a slave axis.
The instruction can generate following system errors:

- "4101 - Inconsistent axis management" when **[axis]** is executing a synchronized movement or a multilinear interpolation or an ISO movement.
- "4105 - instruction not executable on axis" when **[axis]** is a counting axis.
- "4399 - parameter out of range" if the **[time]** indicated is equal or less than 0.

ENDMOV

Syntax

ENDMOV	**axis [, position]**

Arguments

axis	name of axis device
position	constant or variable.

Description

It stops movement of the specified axis. The difference from the STOP instruction is that when movement is interrupted it can not be restarted by using the START instruction. If the parameter **position** is specified, you can set the position at which the axis will end its movement, otherwise the point at which the axis stops will depend on current speed and the last programmed deceleration. Where necessary, to reach the end-of-movement point, the controller reverses axis motion.

Note

This parameter is used only if the movement concerns a point-to point movement. In case of interpolated movement, the movement of the axis stops without considering the **position** value.

Example

```
; stops current movement, taking axis to 0.0 position
ENDMOV    X, 0.0
```

ENABLECORRECTION

Syntax

ENABLECORRECTION	**axis [, axis1, ..., axis6]**

Arguments

axis	name of axis device
axis1, ..., axis6	name of axis device

Description

Enables the linear correction for the specified **axis**. The correction consists of the auto-correction and the crossed correction. The auto-correction is a correction of the real position of an axis in relation to its own position, a crossed correction is a correction of the real position of an axis in relation to the position of other axes. Up to five crossed correctors can be defined.
The first parameter is the axis whose correction is to be deactivated, if it is the only parameter specified all the corrections present in the configuration are activated.
The following parameters allow the specification of which corrections are to be activated, if one of

these coincides with the first parameter the auto-correction is activated.
See also DISABLECORRECTION.

NOTE: For the instruction to have effect the correction must also be enabled in the configuration.

Example

```
; enables all the corrections contained in the configuration for axis X
ENABLECORRECTION      X

; enables only the auto-correction for axis X
ENABLECORRECTION      X, X

; enables the auto-correction and
; the crossed correction (towards X and Y) for axis Z
ENABLECORRECTION      Z, X, Y, Z
```

FASTREAD

Syntax

FASTREAD **axis1, state, variable1 [,axis2, variable2],[…, axis8, variable8]**

Arguments

axis1…[…axis8]	name of axis devices. Axis1 is the master axis
state	predefined constant. It can assume the following values: **ON** rising edge **OFF** falling edge
variable1…[…variable8]	variable or double matrix/vector element. Memorised position

Description

The positions of the indicated **axes** are read and saved in the **variables** the instant the rapid input of **axis1** (Master axis) switches to the set state.

If the indicated axes are analog, they must be part of the same board (8 for ALBN and 4 for TRS-AX).
If the indicated axes are digital, the rapid input signal is located directly on the drive; therefore, in case of multiple fastread, the signal should be connected in parallel on various devices.
If the indicated axes are configured on EtherCAT bus, they must be part of the same drive.
The instruction ends when the input switches to the indicated **state (ON/OFF)**.
If a STOP instruction is executed before switching to rapid input, these instructions remain active and restart after the START instruction.
More than one fast reading can be activated at the same time on the same axis board.

During the execution of the instruction it is not possible to execute the instructions SETPZERO and SETPFLY at the same time on the same axis, if it is connected to boards with Mechatrolink II bus.

Note

The rapid input for the axes being part of an ALBNT board stands on the **axis1** connector and doesn't need to be configured in virtual-physical
The rapid input for digital axes on board with Mechatrolink II bus stands on **EXTI2 input** and doesn't need to be configured in virtual-physical. The rapid inputs of digital Mechatrolink II axes need to be "short circuited", because the axis coordinate should stored only with reference to its own rapid axis.

FREE

Syntax

FREE **axis [, voltage]**

Arguments

axis	name of axis device
voltage	constant float or variable float. Reference voltage

Description

It sets the **axis** in "open loop" (Free) mode, disabling the *position control*. If the voltage parameter is specified, the axis reference voltage is set on the specified value.

This instruction can be used in the case of measuring axes, for position detection, or for axes whose movement can be forced by external mechanical instruments which could alter their position.
During functioning the position of the axis is regularly detected and updated, allowing to position the axis definitively after enabling position control (instruction NORMAL).

HELICABS

Syntax

HELICABS **[label],axis1, position1, axis2, position2, axis3, position3, direction, ±radius [,angle [, numrev [, axis4, position4 [, ..., axis6, position6]]]]**

Arguments

label	constant or variable integer. Label identifying a displacement bloc
axis1...axis3[...axis6]	name of axis devices
position1...position3[... position6]	constant or variable. Absolute move position
direction	integer variable. Kind of rotation clockwise/anticlockwise (CW/CCW)
radius	constant or variable. Radius of the helix
angle	constant or variable. Angle of starting point
numrev	constant or variable. Number of revolutions

Description

Helicoidal interpolation with absolute move equal to programmed positions **position1**, **position2** and **position3**. The movement consists in a circular interpolation associated to axes **axis1** and **axis2** (using the same syntax rules as CIRCABS /CIRCINC, relative to the arguments **direction**, **±radius** and **angle)**, and an associated linear of axis3 (and possibly **axis4**, **axis5** and **axis6**). The helicoidal movement can be developed in a series of revolutions, as indicated by the argument **numrev**. The position of the axis with linear movement (as the possible positions of **axis4**, **axis5** and **axis6**) refers to the total move (not to move/revolution). The optional parameter **label** is used in association with the instruction SETLABELINTERP to indentify univocally the displacement bloc.
Step-by-step axes can only be used in this instruction, if they are controlled by a TRS-AX remote. In this case it must be taken into account that the word interpolation refers to a coordinated movement of more axes affected by discrete error due to axis piloting method.

Note

1. Contornature condition is evaluated only on the three first axes making up the reference system. Adding and possibly modifying a further one, you obtain an incorrect management of the speed profile. To obtain a correct movement, an instruction WAITSTILL between an instruction HELICABS and the other should be interposed.
2. If a reference local system is set using the instruction SETRIFLOC the three axes definint the new reference system should be always be indicated among the parameters of the instruction HELICABS, even if they do not displace anything.

HELICINC

Syntax

HELICINC **[label],axis1, position1, axis2, position2, axis3, position3, direction, ± radius [,angle [, numrev [, axis4, position4 [, ..., axis6, position6]]]]**

Arguments

label	constant or variable integer. Label identifying a displacement bloc
axis1...axis3[...axis6]	name of axis devices
position1...position3[...position6]	constant or variable. Incremental move position
direction	integer variable. Type of rotation clockwise/anticlockwise (CW/ CCW)
radius	constant or variable. Radius of the helix
angle	constant or variable. Angle of starting point
numrev	constant or variable. Number of revolutions

Description

Helicoidal interpolation with incremental move equal to programmed positions **position1**, **position2** and **position3**.
The movement consists in a circular interpolation involving axes **axis1** and **axis2** (using the same syntax rules as CIRCABS /CIRCINC, relative to arguments **direction**, **±radius** and **angle)**, and a

linear interpolation involving **axis3** (and possibly **axis4**, **axis5** and **axis6**).
The helicoidal movement can be developed in a series of revolutions as indicated by the argument **numrev**.
The position of the axis with linear movement (as the possible positions of **axis4**, **axis5** and **axis6**) refers to the total move (not to move/revolution). The optional parameter **label** is used in association with the instruction SETLABELINTERP to indentify univocally the displacement bloc.
Step-by-step axes can only be used in this instruction, if they are controlled by a TRS-AX remote. In this case it must be taken into account that the word interpolation refers to a coordinated movement of more axes affected by discrete error due to axis piloting method.

Note

1. Contornature condition is evaluated only on the three first axes making up the reference system. Adding and possibly modifying a further one, you obtain an incorrect management of the speed profile. To obtain a correct movement, an instruction WAITSTILL between an instruction HELICINC and the other should be interposed.
2. If a reference local system is set using the instruction SETRIFLOC the three axes that define the new reference system should be always be indicated among the parameters of the instruction HELICINC, even if they do not displace anything.

JERKCONTROL

Syntax

JERKCONTROL axis, state

Arguments

axis	name of axis devices
state	predefined constant. It can assume the following values: **ON** rising edge **OFF** falling edge

Description

According to whether the parameter **state** is set on ON or OFF, it enables or disables the jerk control on **axis** interpolation and point-to-point movements.The jerk control is enabled only with axes that have configured one acceleration ramp and Esse deceleration. If the axis has configured one Linear ramp the jerk is not checked.

JERKSMOOTH

Syntax

JERKSMOOTH axis, value

Arguments

axis	name of devices of axis type.
value	constant or variable float.

Description

In any classic interpolated movements, the axes can move while contouring, that is without stopping between a bloc and the next one. This occurs, if discontinuous function of tangency in the blocs is lower than the value "Maximum contouring angle", set in the module configuration (default value is 15), or lower than the value set through the instruction SETCONTORNATURE. In the opposite case, the axes are stopped in the edge point with controlled deceleration and let start again along the new bloc with controlled accelerations and speed rates. However, stop and restart reduce the machine movement performances. When the contouring angle takes on consistent values such as, e.g., a discontinuous function of tangency value higher than 1 degree, remarkable jumps of speed for the axes involved in contouring are determined, with infinite acceleration values and discontinuous functions in the speed rate profile, consequently. According to a value established by the user, the instruction JERKSMOOTH allows to link smoothly, that is with acceleration and speed continuity, the speed profiles of the axis while contouring. It should be noted that this smooth link inserts little variation in the performed trajectory compared to the performed one, because around the contouring point the axes show a speed rate profile different from the theoretical one.
The variable **value** expressed through a percentage value between 0 and 100, defines how much the speed rates profiles should be smoothly linked. A value equal to 0 maintains a theoretical profile by creating some discontinuities in the accelerations and in the speed rates profiles. A value equal to 100 obtains smooth linked profiles, a better performance, but also the high deviation from the theoretical trajectory, proportionate to the speed rate along the trajectory.

Note

The instruction is only applied in the movements with classic interpolation (instructions LINEARABS, LINEARINC, CIRCABS, CIRCINC, HELICABS, HELICINC). It cannot be applied in movements of multiaxis interpolation (instruction MULTIABS and MULTIINC).

LINEARABS

Syntax

LINEARABS **[label],axis1, position1, [axis2, positon2 [, axis3, position3 [, ..., axis6, position6]]]**

Arguments

label	constant or variable integer. Label identifying a displacement bloc
axis1[...axis2[...axis6]]	name of axis devices
position1[...position2[...position6]]	constant or variable. Absolute move position

Description

Linear interpolation, with absolute move, in positions specified by position1, position2, etc. The optional parameter label is used in association with the instruction SETLABELINTERP to identify univocally the displacement bloc.

Step-by-step axes can only be used in this instruction, if they are controlled by a TRS-AX remote. In this case it must be taken into account that the word interpolation refers to a coordinated movement of more axes affected by discrete error due to axis piloting method.

Note

1. Contornature condition is evaluated only on the three first axes making up the reference system. Adding and possibly modifying a further one, you obtain an incorrect management of the speed profile. To obtain a correct movement, an instruction[****]WAITSTILL between an instruction LINEARABS and the other should be interposed.
2. If a reference local system is set using the instruction SETRIFLOC, the three axes that define the new reference system should be always be indicated among the first three parameters of the instruction LINEARABS, even if they do not displace anything.

LINEARINC

Syntax

LINEARINC **[label],axis1, position1, [axis2, positon2 [, axis3, position3 [, ..., axis6, position6]]]**

Arguments

label	constant or variable integer. Label identifying a displacement bloc
axis1[...axis2[...axis6]]	name of axis devices
position1[...position2[...position6]]	constant or variable. Incremental move position

Description

Linear interpolation, with *incremental move*, in positions specified by **position1**, **position2**, etc. The optional parameter **label** is used in association with the instruction SETLABELINTERP to identify univocally the displacement bloc.

Step-by-step axes can only be used in this instruction, if they are controlled by a TRS-AX remote. In this case it must be taken into account that the word interpolation refers to a coordinated movement of more axes affected by discrete error due to axis piloting method.

Note

1. Contornature condition is evaluated only on the three first axes making up the reference system. Adding and possibly modifying a further one, you obtain an incorrect management of the speed profile. To obtain a correct movement, an instruction WAITSTILL between an instruction LINEARABS and the other should be interposed.
2. If a reference local system is set using the instruction SETRIFLOC, the three axes that define the new reference system should be always be indicated among the parameters of the instruction LINEARINC, even if they do not displace anything.

MOVABS

Syntax

MOVABS	**axis1, value1 [, axis2, value2 [, …, axis6, value6]]**

Arguments

axis1…[…axis6]	name of axis devices
value1…[…value6]	constant or variable. Value of absolute move

Description

It instructs the specified axes to execute an *absolute moveme* according to values specified in **value1 [,…value6]**.
To execute the move the axis must not be engaged in an interpolated move and it must be in position or in window. The movement of the axis begins as soon as the instruction is executed. If more than one point-to-point movement instruction is performed in the same task, they are chained. If a second task tries to carry out point-to-point instructions on an axis that is already engaged in a move, this task will wait for the move commanded by the first task to end.
It is also possible to change the velocity of a point-to-point movement and the following move using the instruction SETVEL. The two movements will be linked by a speed ramp without stopping the axes.
If the instruction SETVEL is not used, the highest possible velocity is represented by the value of the manual speed configured.
A point-to-point movement can be halted with the instruction STOP and subsequently restarted with the instruction START. During the interruption of the movement the axis remains in a normal running state even though physically it is not moving.
A move can be aborted with the instruction ENDMOV. In this case it cannot be restarted.

NOTE: Previously point-to-point movements:

- allowed no speed variation unless the axis was motionless. The current behaviour is similar to that of interpolated movements.
- when interrupted by a STOP the corresponding axis assumed the state "in position".

2) We suggest the reader to use linear interpolation instructions instead of point-to-point movement instructions, when the number of moving blocks exceeds 32 and the blocks are made by micro-segments. For further details references shall be made to the document ""Limiti Firmware Movimento Punto Punto.doc" available from T.p.A. S.p.A.

Example

Homing Routine on Interrupt

Example 2

```
; speed change
Function SpeedChange
    setvel     X, 20
    setvel     X, 20
    movabs     X, 100, Y, 200
    movabs     X, 150, Y, 180
    setvel     X, 5
    movabs     X, 80, Y, 100
    waitstill  X, Y
fret
```

MOVINC

Syntax

MOVINC	**axis1, value1 [, axis2, value2 [, …, axis6, value6]]**

Arguments

axis1…[…axis6]	name of axis devices
value1…[…value6]	constant or variable. Value of incremental move

Description

It instructs each axis to execute an *incremental move* on the basis of the corresponding **value**.
To execute the move the axis must not be engaged in an interpolated move and it must be in

position or within tolerance. The movement of the axis begins as soon as the instruction is executed. If more than one point-to-point movement instructions on the same task is executed, they are chained. If a second task tries to carry out point-to-point instructions on an axis that is already engaged in a move, this task will wait for the move commanded by the first task to end.
It is also possible to change the speed of a point-to-point movement and the following move using the instruction SETVEL. The two movements will be linked by a speed ramp without stopping the axes.
If the instruction SETVEL is not used, the highest possible speed is represented by the value of the manual speed configured.
A point-to-point movement can be halted with the instruction STOP and subsequently restarted with the instruction START. During the interruption of the movement the axis remains in a normal running state even though physically it is not moving.
A move can be aborted with the instruction ENDMOV. In this case it cannot be restarted.

NOTE: Previously point-to-point movements:

- allowed no speed variation unless the axis was motionless. The current behaviour is similar to that of interpolated movements.
- when interrupted by a STOP the corresponding axis assumed the state "in position".

2) We suggest the reader to use linear interpolation instructions instead of point-to-point movement instructions, when the number of moving blocks exceeds 32 and the blocks are made by micro-segments. For further details, references shall be made to the document "Limiti Firmware Movimento Punto Punto.doc" available from T.p.A. S.p.A.

Example

Homing Routine of an axis

Example 2

```
; speed change
Function SpeedChange
    setvel     X, 20
    setvel     X, 20
    movinc     X, 100, Y, 200
    movinc     X, 150, Y, 180
    setvel     X, 5
    movinc     X, 80, Y, 100
    waitstill  X, Y
fret
```

MULTIABS

Syntax

MULTIABS	**[label],axis1, value1, [axis2, value2 [, axis3, value3 [,…, axis16, value 16]]]**

Arguments

label	constant or variable integer. Label identifying a displacement bloc
axis1 … axis16]	name of axis devices
value1… [… value16]	constant or variable. Value of theoretical position of displacement bloc end

Description

Absolute multi-linear interpolation up to 16 axes. This interpolation movement enables to advance the speed profiles, properly setting their respective tolerances on the axes by means of the instruction SETTOLERANCE (axes tolerance refers to a portion of path, where a constant interpolation ratio could not possibly exist). Axes addition order into the MULTIABS instruction **should** always be the same and **all** the axes involved in the movement should be present. The move blocs are queued in the normal lookahead and the movement is partially joined to the execution of an instruction WAITSTILL, STARTINTERP or to the filling of the same lookahead. From the axes involved in the move one can be used as a collider by means of the WAITCOLL instruction. The optional parameter **label** is used in association with the instruction SETLABELINTERP to identify univocally the displacement bloc.
Step-by-step axes can only be used in this instruction, if they are controlled by a TRS-AX remote. In this case it must be taken into account that the word interpolation refers to a coordinated movement of more axes affected by discrete error due to axis piloting method.

Note

With this kind of interpolation, virtual reference systems (SETRIFLOC and RESRIFLOC instructions) cannot be used. It is possible to perform some movements with chained axes (in CHAIN). The axes involved in the multiaxis interpolated movement should be declared master of other axes not involved in the movement. Furthermore, FeedRateOvveride can be applied.

Example

```
SETQUOTE        x, 0
SETQUOTE        y, 0
SETQUOTE        z, 0

;first bloc
setveli         x, velx1
setveli         y, vely1
setveli         z, velz1
multiabs        x, positionx1,  y,positiony1,  z,position1

;second bloc
settolerance    x, tollx2,  y,tolly2,  z,tollz2
setveli         x, velx2
setveli         y, vely2
setveli         z, velz2
multiabs        x, positionx2, y,positiony2, z,positionz2

;third bloc
settolerance    x, tollx3,  y,tolly3,  z,tollz3
setveli         x, velx3
setveli         y, vely3
setveli         z, velz3
multiabs        x, positionx3,  y,positiony3, z,positionz3

; fourth bloc
settolerance    x, tollx4,  y,tolly4,  z,tollz4
setveli         x, velx4
setveli         y, vely4
setveli         z, velz4
multiabs        x, positionx4, y,positiony4, z,positionz4
waitstill       x, y,z
```

MULTINC

Syntax

MULTIINC **[label],axis1, value1, [axis2, value2 [, axis3, value3 [,…, axis16, value 16]**

Arguments

label constant or variable integer. Label identifying a displacement bloc

axis1 … axis16] name of axis devices

value1… [… value16] constant or variable. Value of theoretical position increase of displacement bloc end

Description

Incremental multi-linear interpolation up to 16 axes. This interpolation movement enables to advance the speed profiles, properly setting their respective tolerances on the axes by means of the instruction SETTOLERANCE (axes tolerance refers to a portion of path, where a constant interpolation ratio could not possibly exist). Command of axes addition into the MULTINC instruction **should** always be the same and **all** the axes involved in the movement should be present. The movement blocs are queued in the normal lookahead and the movement is partially joined to the execution of an WAITSTILL, STARTINTERP instruction or to the filling of the same lookahead. From the axes involved in the movement one can be used as a collider by means of the WAITCOLL instruction. The optional parameter **label** is used in association with the instruction SETLABELINTERP to identify univocally the displacement bloc.

Step-by-step axes can only be used in this instruction, if they are controlled by a TRS-AX remote. In this case it must be taken into account that the word interpolation refers to a coordinated movement of more axes affected by discrete error due to axis piloting method.

Note

With this kind of interpolation, virtual reference systems (SETRIFLOC and RESRIFLOC instructions) cannot be used. It is possible to perform some movements with chained axes (in CHAIN). The axes involved in the multiaxis interpolated movement should be declared master of other axes not involved in the movement. Furthermore, FeedRateOvveride can be applied.

NORMAL

Syntax

NORMAL	**axis**

Arguments

axis	name of axis device

Description

It enables the *position control* on the **axis** and disables the *axes chain*.
When the system is switched on, all the configured axes set in free state and switch to normal state when this instruction is executed or when the first movement takes place.
However it is advisable to execute this instruction before carrying out axis reset procedure, to restore any existing emergency conditions.

RESRIFLOC

Syntax

RESRIFLOC	**axis1, axis2, axis3**

Arguments

axis1...axis3	name of devices. Type of axis

Description

It resets the absolute reference system for axes X Y Z (**axis1**, **axis2**, **axis3**).
It is normally used after setting a rototranslation reference system with a SETRIFLOC instruction.

SETINDEXINTERP

Syntax

SETINDEXINTERP	**axis, varname**

Arguments

axis	name of axis device
varname	name of global integer variable

Description

It defines an index which counts the number of interpolation blocks executed by an axis.
During interpolation movements, the variable **varname** increases by 1 at each block change.

Note

The variable used as index must be a group global variable or a machine global.

SETLABELINTERP

Syntax

SETLABELINTERP	**axis, value**

Arguments

axis	name of device of type axis
value	global variable of type integer

Description

In the variable **value** during a movement in interpolation, every time that the bloc is changed, the label value of the new bloc is assigned. The label is defined in the instructions of interpolated movement.

Note

The variable **value** should be a global group variable or a machine global.

SETPFLY

Syntax

SETPFLY	**axis, state, speed, position,[error]**

Arguments

axis	name of axis device
state	predefined constant. It indicates the state of the micro to be tested. Acceptable values are: **ON** **OFF**
speed	float constant or variable
position	constant or variable
error	integer variable. Error code

Description

It allows you to reset the **axis** position "on the fly". The resetting is piloted by a switch connected to the rapid input of the axis connector (on boards with Mechatrolink II bus reference is made to EXTI1).
During **axis** movement, it waits for the corresponding home micro to switch to the indicated **state**. When this transition is intercepted, the real position of the axis is reset on zero, without interrupting movement, and target **position** and **speed** are automatically and dynamically redefined. If the set position is reached without detecting an input change and the parameter **error** has not been set, a system error is generated. If an **error** parameter has been set, this will contain the numeric code for the corresponding system error.
In this case the homing has not been executed and it is necessary to execute the SETQUOTE instruction to reset the micro search.
To interrupt the "on the fly" homing execution, execute a NORMAL on the axis or simply end the task that requested the homing.

During the execution of the instruction it is not possible to execute the instructions SETPZERO and FASTREAD at the same time on the same axis, if it is connected to boards with Mechatrolink II bus.

Example

Homing Routine on Interrupt

SETPFLYCHAINSTRAT

Syntax

SETPFLYCHAINSTRAT	**axis, type**

Arguments

axis	name of axis device
type	Integer constant. Permitted values: 0 = only the master axis zeroes the coordinate. The slave axis keeps the previous coordinate not equal to 0 = both master and slave axes synchronously zero the coordinate

Description

This instruction enables to set, how the indicated slave axes will behave for a master setpfly instruction.
The istruction has to be executed indicating the slave axis. If the variable **Type** is omitted, a default value equal to 0 is set.

SETPZERO

Syntax

SETPZERO **axis, position [,error]**

Arguments

axis	name of axis device
position	constant or variable. It is an incremental position
error	integer variable. Error code

Description

It starts an incremental movement of the **axis** in the specified **position** and waits for the encoder zero pulse to be detected (before reaching the specified position).
As soon as the pulse is detected the real position is set on zero and the axis is stopped.
If the set position is reached without detecting the Zero pulse and the parameter **error** has not been set, a system error is generated. If an **error** parameter has been set, this will contain the numeric code for the corresponding system error. In this case the set point has not been executed and the SETQUOTE instruction must be executed to reset the pulse search.
The incremental position can reach a maximum of 50,000 encoder impulses.
The instruction is not applicable to digital axes on ALBSLM.
The movement of the axes, generated by this instruction, can be interrupted with a STOP and restarted by a START.
If the instruction is executed with S-CAN axes and with EtherCAT axes, a FREE instruction must be executed first.

During the execution of the instruction it is not possible to execute the instructions SETPZERO and FASTREAD at the same time on the same axis, if it is connected to boards with Mechatrolink II bus.

Example

```
Function TestSetpZero
   free       X
   setpzero   x,100
fret
```

SETQUOTECHAINSTRAT

Syntax

SETQUOTECHAINSTRAT **axis, [value]**

Arguments

axis name of axis device

value integer variable. Permitted values:
0= only the master axis zeroes the new coordinate, the slave axis keeps the previous coordinate
not equal to 0= the coordinates of the slave axes are synchronously initialized with the master axis coordinates

Description

This instruction enables to set how the indicated slave axis will behave for a master SETQUOTE instruction
The instruction has to be executed on the slave axis.
If the variable **value** is omitted, a default value equal to 0 is set.

SETPZEROCHAINSTRAT

Syntax

SETPZEROCHAINSTRAT **axis, [value]**

Arguments

axis name of axis device

value integer variable. Permitted values:
0= only the master axis zeroes the coordinate, the slave axis keeps the previous coordinate
not equal to 0= both master and slave axes synchronously zero the coordinate

Description

This instruction enables to set how the indicated slave axis will behave for a master SETPZERO instruction
The instruction has to be executed on the slave axis.
If the variable **value** is omitted, a default value equal to 0 is set.

SETQUOTE

Syntyax

SETQUOTE **axis, position**

Arguments

axis	name of axis device
position	constant or variable

Description

This instruction forces, at the same time, the theoretical and the real position of an axis to the value specified in **position**. If the axis is moving, this instruction causes the axis to stop abruptly as it is suddenly set in position (real quote coincides with target quote). For this reason we do not recommend using this instruction on moving axes if not at a very reduced speed.

Example

Axis Homing routine

SETRIFLOC

Syntax

SETRIFLOC **position1_ax1, position2_ax1, position3_ax1,**
position1_ax2, position2_ax2, position3_ax2,
position1_ax3, position2_ax3, position3_ax3,
axis1, axis2, axis3

Arguments

position1ax1...position3_ax3	director cosine of the three axes
axis1...axis3	name of devices. Axes

Description

It allows you to activate an X' Y' Z' Cartesian reference system with a rototranslation with respect to the X Y Z absolute reference system of the machine, represented by the physical axes **axis1**, **axis2** and **axis3**.
The nine arguments indicate the Director Cosines of the three local axes in reference to the absolute axes

$\cos\alpha_1$	$\cos\beta_1$	$\cos\gamma_1$
$\cos\alpha_2$	$\cos\beta_2$	$\cos\gamma_2$
$\cos\alpha_3$	$\cos\beta_3$	$\cos\gamma_3$

which compose the transformation matrix of the coordinates.
The origin of the new reference system is set in the current point.
All the interpolation movement instructions, involving axes X, Y and Z, refer to this reference system, until the RESRIFLOC instruction is executed.

SETTOLERANCE

Syntax

SETTOLERANCE **axis1, value1, [axis2, value2 [, axis3, value3 [, ..., axis16, value 16]]]**

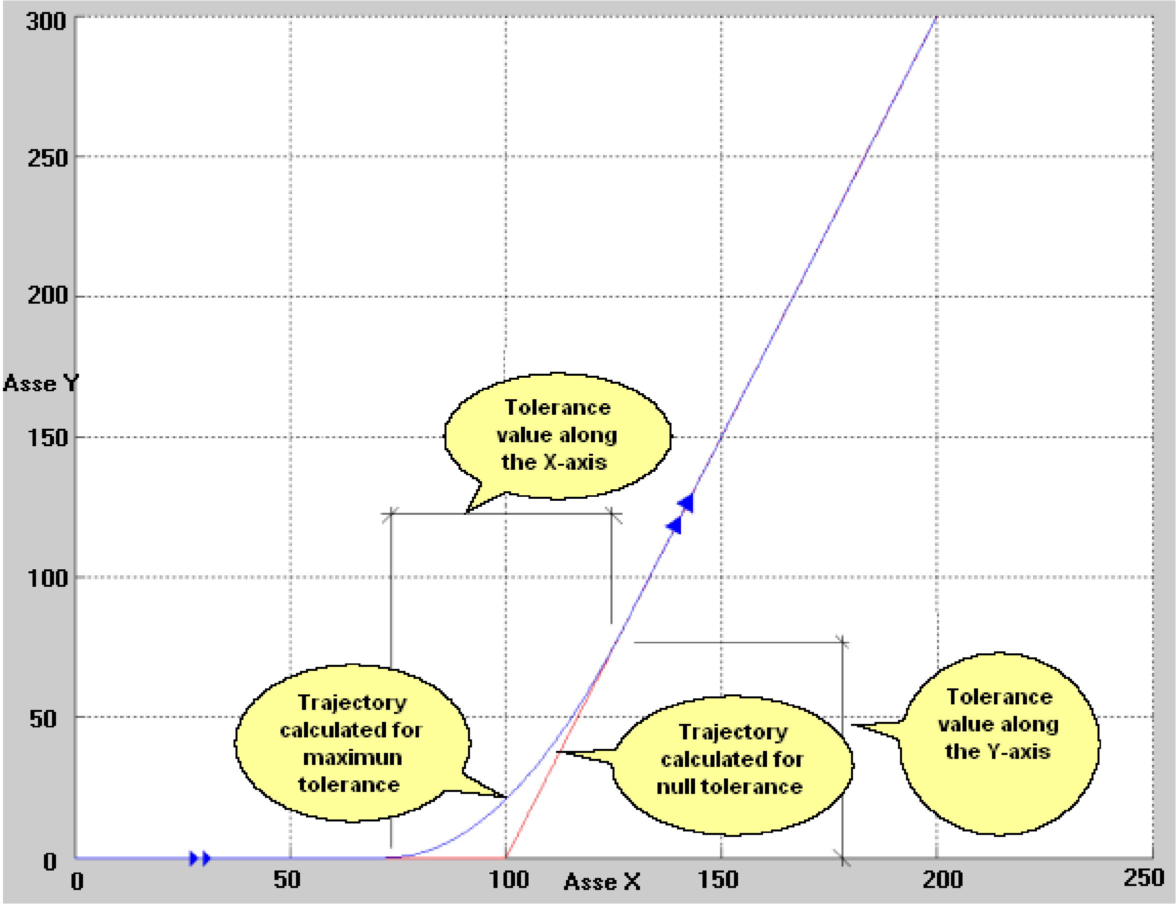
Arguments

axis1...axis16	name of axis devices
value1...[...value16]	constant or variable. Maximum tolerance value that can be applied to the axis.

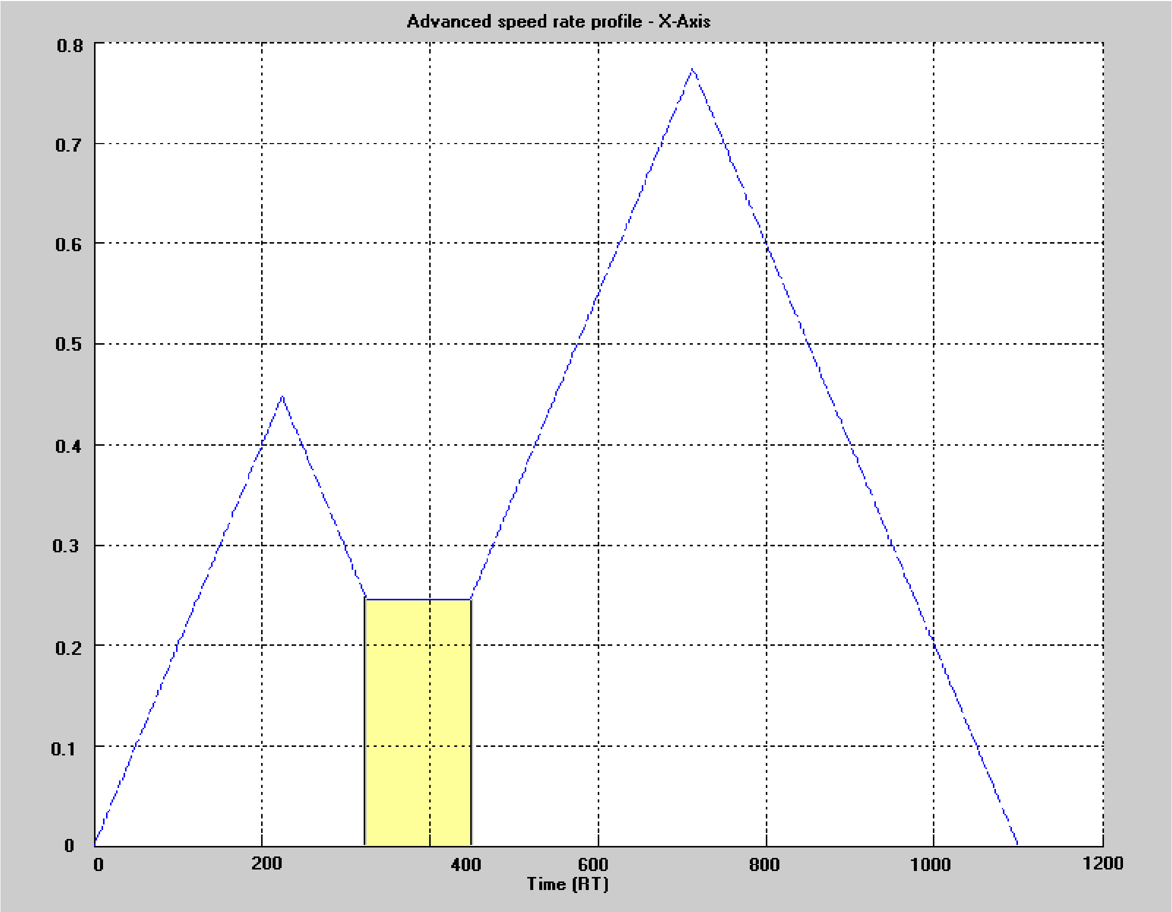
Description

For each defined **axis** it sets the tolerance **value** to apply on the multi-axis interpolation motion.
Tolerance value is the displacement value according to which the axis moves away from the original trajectory in a multi-axis interpolation.

Tolerance has to be set for each **axis** involved in the interpolation and the system will advance the speed rate profiles and respect the tolerances on all the axes without exceeding the ramp space, that represents the upper limit to anticipate the profiles. A missing assignment of tolerance before a multi-axis instruction means that the last tolerance will be applied on the axis itself. If a tolerance value has never been assigned before, the same is considered with null tolerance. In this case each multi-axis motion, involving that axis, does not set any ramp in advance.



A classic multi-axis trajectory is shown above and is made of two moving blocks, where the first one consists in a displacement of 100 of the X-axis, while the second one consists in an Y-axis motion of 300 and in an X-axis motion of 100. The red line marks the trajectory in case of null tolerance, the blue one instead is the trajectory in case of maximum tolerance axis.
The tolerance can also be seen as the area subtended by the speed rate profile during the time of advance, as below.



START

Syntax

START **axis**

Arguments

axis name of axis device

Description

It restarts **axis** movement after a **stop**.

STARTINTERP

Syntax

STARTINTERP **axis**

Arguments

axis name of axis device

Description

It starts an interpolation whose channel is identified by **axis**. Normally the movement of axes associated to an interpolation channel begins when the interpolation buffer is completely full (512 instructions) or when a WAITSTILL instruction is given, to stop movement. This allows the algorithm of the interpolator to determine optimal speed profiles, as it is provided with information concerning a large number of (or all) stages of interpolation movement.
The STARTINTERP instruction allows you to force axis movement even if the above described conditions are not fulfilled.

STOP

Syntax

STOP	**axis**

Arguments

axis	name of axis device

Description

It interrupts **axis** movement. The axis executes a deceleration ramp whose length depends on current speed and configuration parameters.

Example

Homing Routine of an axis

SWITCHENC

Syntax

SWITCHENC	**axis1, [axis2, [direction, coordinate]]**

Arguments

axis1	name of the device of axis type
axis2	name of the device of axis type indicates counting axis
direction	predefined constant. **UP**=encoder exchange, when the coordinate in positive direction is exceeded **DOWN**=encoder exchange, when the coordinate in negative direction is exceeded
coordinate	constant or double variable

Description

Allows you to replace the encoder of **axis 1** with the encoder of **axis2**. The encoder is exchanged when the quote indicated is exceeded in positive (UP) or in negative (DOWN) **direction**. If the parameters **direction** and **coordinate** are left out, the encoder exchange is immediately executed, regardless of the axis position.

If only axis1 is **declared,** the functioning with a single encoder is restored.

Axis1 cannot be of step-by-step, counting and virtual type, **Axis2** can be a counting axis only. Further, both **axis1** and **axis2** cannot be involved in movements in chain as slave axis

The instruction generates the system error 4101 - Inconsistent axis AxisName management", when **axis1** or **axis2** are declared as slave in a movement in chain and the system error 4105 Instruction not executable on axis AxisName, when the declared axis type does not belong to the possible ones.

SYNCROOPEN

Syntax

SYNCROOPEN	**channel, deltaT, matrix, var, [smoothing]**

Arguments

channel	variable or constant integer. Number of channel to open
deltaT	time interval
matrix	matrix in which the positions are saved
var	number of lines generated
smoothing	optional, admissible values: **ON** and **OFF**. Enables the profile smoothing function

Description

Opens a synchronized movement channel. With this type of movement a points profile is generated that is then executed using the instruction COORDIN. The profile is generated starting from a series of "crossing points" through which the axes must move all at the same moment. The crossing points are assigned using the instruction SYNCROMOVE.
The parameter **deltaT** assigns frequency with which the points of the profile joining the various crossing points are generated.
The parameter **matrix** specifies the GPL matrix in which the profile of points is saved. The matrix

must be composed of a double (or float) column for each axis involved in the movement, plus an integer column (the last). The double column is used to save the positions of the profile points, while the integer column is used to save an index which corresponds to the crossing point to which the current stroke of the profile must be brought. This index is useful for synchronizing the movement of the axes with other activities (e.g. activating an output).
The parameter **var** is a variable in which the number of points generated is saved; it is used by the instruction COORDIN to know what the last useful line of the GPL matrix is in which the profile is saved.
The parameter **smoothing** allows the activation of a function that avoids discontinuities in the speed of the axes, thus making the movement more fluid.
Within a profile the speed and the acceleration of the individual axes involved can be varied and the speed of the axes can be scaled down, see SYNCROSETACC, SYNCROSETDEC, SYNCROSETFEED.
Once all the crossing points are assigned the profile can be generated using the instruction SYNCROSTARTMOVE.
When the generation of the profile is over the synchronized movement channel is closed with the instruction
SYNCROCLOSE.
Up to 4 synchronized movement channels can be opened at the same time. The channel enumeration starts from 1.

Example

Synchronized movements

SYNCROCLOSE

Syntax

SYNCROCLOSE	**channel**

Arguments

channel	variable or constant integer. Number of channel to close

Description

Closes a synchronized movement channel. See SYNCROOPEN.

Example

Synchronized movements

SYNCROMOVE

Syntax

SYNCROMOVE	**channel, axis1, position1, [axis2, position2, [... axis32, position32]]**

Arguments

channel	variable or constant integer. Number of synchronized movement channel
axis1	first axis
position 1	position of first axis
axis32	thirty second axis
position32	position of thirty second axis

Description

This assigns a crossing point for the axes and the movement channel specified. The crossing points are inserted in a internal table that is processed when the instruction SYNCROSTARTMOVE.
is executed.
The instruction allows up to thirty two axes to be moved.
Step-by-step axes can be used in this instruction, only if controlled by a TRS-AX remote. In this case it must be taken into account that the word interpolation refers to a coordinated movement of more axes affected by discrete error due to axis piloting method.
The instruction **SYNCROMOVE** must be preceded by the instruction to open the synchronized movement channel: SYNCROOPEN.

Example

Synchronized movements

SYNCROSETACC

Syntax

SYNCROSETACC	**channel, axis, accPos, accNeg**

Arguments

channel	variable or constant integer. Number of channel
axis	axis device
accPos	variable or constant integer. Acceleration in positive direction
accNeg	variable or constant integer. Acceleration in negative direction

Description

Sets the duration of the acceleration ramps of an axis during a synchronized movement. The acceleration of a movement can be defined in a positive and in a negative direction. See SYNCROOPEN.

Example

Synchronized movements

SYNCROSETDEC

Syntax

SYNCROSETDEC	**channel, axis, decPos, decNeg**

Arguments

channel	variable or constant integer. Number of channel
axis	axis device
decPos	variable or constant integer. Deceleration in positive direction
decNeg	variable or constant integer. Deceleration in negative direction

Description

Sets the duration of the deceleration ramp of an axis during a synchronized movement. The deceleration of a movement can be defined in a positive and in a negative direction. See SYNCROOPEN.

Example

Synchronized movements

SYNCROSETVEL

Syntax

SYNCROSETVEL	**channel, axis, velPos, velNeg**

Arguments

channel	variable or constant integer. Number of channel
axis	axis device
velPos	variable or constant double. Speed in positive direction
velNeg	variable or constant double. Speed in negative direction

Description

Sets the speed of an axis during a synchronized movement. The speed of a movement can be defined in a positive and in a negative direction. See SYNCROOPEN.

Example

Synchronized movements

SYNCROSETFEED

Syntax

SYNCROSETFEED	**channel, axis, feed**

Arguments

channel	variable or constant integer. Number of channel
axis	axis device

feed	variable or constant integer. Feed rate value

Description

Allows the scaling of the speed of all the axes associated with a synchronized movement channel. The parameter **feed** represents the percentage of the speed previously programmed to be applied to the following movements. Admissible values range from 0 to 100.
NOTE: This instruction has effect during the generation of the profile and not during its execution which is triggered by the instruction COORDIN. To scale the speed of the axes while they are in motion use the instruction SETFEEDI.

Example

Synchronized movements

SYNCROSTARTMOVE

Syntax

SYNCROSTARTMOVE	**channel, [line]**

Arguments

channel	variable or constant integer. Number of channel
line	initial process line

Description

Start the processing of the profile for the specified channel. The optional parameter **line** allows the specification of which will be the first line of the internal table in which the SYNCROMOVE instructions to include in the processing are queued; if this is omitted the processing starts from the first line. See SYNCROOPEN.

Example

Synchronized movements

WAITCOLL

Syntax

WAITCOLL	**axis, value, timeout, delta**

Arguments

axis	name of the axis device
value	constant or variable. Absolute position value
timeout	constant or variable. It is the waiting time, when the axis is still
delta	constant or variable.it is the window value to obtain a still axis

Description

When the axis moves, the achievement of a programmed position can be prevented by an obstacle of mechanical nature, represented at times also by the same workpiece. In this case the system generates an error in the system "servoerror" or "movement not finished". This instruction defines a position value at which
- the system begins to verify the presence of a collision;
- the waiting time **(timeout)** before the **axis**, after the collision, is placed on "position";
- the **delta** that defines the tolerance on the axis positioning.

When the axis exceeds its position, which is defined in **value**, the system checks, whether the axis is still moving . Once the obstacle is intercepted, the critical situation is identified and, while ensuring the engine thrust, the loop error exceeding the limit is not checked anymore. The motion direction on which the collision occurred is verified, has the same direction of the last movement joined at the end. The timeout is expressed in seconds, the **delta** value should be greater than 0.001 mm and less than the difference between the programmed arrival position and the position **value**.

The instruction can be used with the multi-axis interpolator, since within such interpolator the temporary loss of the interpolation link.
The instruction can be applied also to virtual axes and to Master axes of a movement in chain.
An error system is generated when:

- the axis is executing a classic interpolated movement (see instructions LINNEARBS, LINEARINC,. CIRCABS, CIRCINC, HELICABS, HELICINC) or in coordinated motion
- the **axis** is a slave-axis

- the **axis** is a counting axis or a step-to-step axis
- the **value** set is higher than the end-movement position

Example

```
; sets the X axis position
SETQUOTE  X, 0.0
; moves the x axis to the absolute position 1000
MOVABS               X, 1000.0
; waits for the collision point, waits 2 seconds before setting
; the axis on "position", after
; intercepting a collision with a precision of 0.01 mm
WAITCOLL             X, 980.0,2.0,0.01
```

WAITDEC

Syntax

WAITDEC **axis1 [, ..., axis6]**

Arguments

axis1 [...,axis6] name of axis device

Description

It waits for the deceleration state or one of the subsequent states of all the specified **axes** (1÷6). The task where the instruction is executed is put on wait status, until the axis reaches the acceleration, coordinate, wait on the higher threshold, wait on the lower threshold and axis quiescent waiting state.

Axis states are identified by an integer:
- acceleration = 1
- steady = 2
- deceleration = 3
- coordinate = 4
- wait on the higher threshold = 5
- axis quiescent waiting = 6
- wait on the lower threshold = 7

WAITREG

Syntax

WAITREG **axis1 [, ..., axis6]**

Arguments

axis1 [...,axis6] name of axis device

Description

It waits for the regime state or one of the subsequent states of all the specified **axes** (1÷6). Il task in cui viene eseguita l'istruzione viene messo in attesa fino a quando l'asse si trova negli stati di steady, deceleration, coordinate, wait on the higher threshold, wait on the lower threshold and axis quiescent waiting state.

Axis states are identified by an integer:
- acceleration = 1
- steady = 2
- deceleration = 3
- coordinate = 4
- wait on the higher threshold = 5
- axis quiescent waiting = 6
- wait on the lower threshold = 7

WAITSTILL

Syntax

WAITSTILL	**axis1 [, ..., axis6]**

Arguments

axis1 [...,axis6]	name of axis device

Description

It waits for all the specified **axes** (1÷6) to end movement (Position state).

Example

Axis Homing routine

WAITTARGET

Syntax

WAITTARGET	**axis1 [, ..., axis6]**

Arguments

axis1 [...,axis6]	name of axis device

Description

It waits for the theoretical current position of all the specified **axes** (1÷6) to reach target position.
The real quote will not match the theoretical position until the loop error is cleared.

WAITWIN

Syntax

WAITWIN	**axis1 [, ..., axis6]**

Arguments

axis1 [...,axis6]	name of axis device

Description

It waits for the thereshold state or one of the subsequent states of all the specified **axes** (1÷6).
he task where the instruction is executed is put on wait status, until the axis reaches the wait on the higher threshold, wait on the lower threshold and axis quiescent waiting state.

Axis states are identified by an integer:
- acceleration = 1
- steady = 2
- deceleration = 3
- coordinate = 4
- wait on the higher threshold = 5
- axis quiescent waiting = 6
- wait on the lower threshold = 7

WAITACC

Syntax

WAITACC	**axis [, ..., axis6]**

Arguments

axis1[...,axis6]	name of axis device

Description

It waits for the acceleration state or one of the subsequent states of all the indicated **axes** (1÷6).
The task where the instruction is executed is put on wait status, until the axis reaches the acceleration state or one of the subsequent states.

Axis states are identified by an integer:
- acceleration = 1

- steady = 2
- deceleration = 3
- coordinate = 4
- wait on the higher threshold = 5
- axis quiescent waiting = 6
- wait on the lower threshold = 7

Axis Parameter

Reading/Writing

DEVICEID

Syntax

DEVICEID	**device, variable**

Arguments

device	name of the device or the device parameter
variable	integer variable receiving the logical address

Description

It writes in **variable** the logical address associated to any kind of device.
This instruction may enable an univocal "key" associated to the device, as an index or a search key in data structures.

GETAXIS

Syntax

GETAXIS	**axis, dataname, varname**
GETAXIS	**axis, dataname1, dataname2, [...,dataname20,] matrix[row]**

Arguments

axis	name of axis device
dataname	predefined constant (See list below). Axis parameter(1÷20)
varname	variable or name of device
row	constant or integer variable. Row number of the matrix
matrix	name of matrix

Description

In the first version the instruction reads one datum (**dataname**) of an axis and saves it in a variable.
In the second version, the instruction reads various data of an axis at the same time (from 1 to 20) and saves it, in the same order as it was requested, in the elements of the specified matrix row.
In this case the number of columns of the matrix must correspond to the number of requested data.
The list reported below includes all the predefined constants that can be assigned to the parameter **dataname**.
The first column is the name of the constant.
The second describes the quantity of the axis read by the instruction.
The third is the format of the datum returned to the variable **varname** or the **matrix[row]**, where:

- **d** means **double**,
- **f** means **float**,
- **i** means **integer**
- **b** means **char**.

If the declaration of the variable, where the data will be memorised, is different from the value returned by the instruction, the compiler changes the datum (cast) to the type requested by the user. Sometimes this implies losing a certain amount of data. For example a double value equal to 12,345, changed into an integer, becomes 12. For this reason we recommend keeping to the requested types when declaring **varname** and **matrix[row]** variables.
The last column describes either the return value or the measuring unit of the relative parameter.
Constants beginning with "_CFG" allow to configuration values, that is the values set when the machine is started.

costant	**description**	**type**	**return value**
_CFGTYPE	Axis typology	i	1=analog,3=stepping

costant	description	type	return value
			motor,4=digital,5=count, 6=frequency/ direction,7=virtual
_CFGUM	unit of measure	i	0=millimetres,1=inches,2 =degrees,3=revol.
_CFGRIS	resolution	d	impulses per _UM
_CFGVMAX	maximum speed	f	m/1' or inch/1' or degrees/1' or rotations/1'
_CFGVMAXD	maximum speed in manual mode	f	m/1' or inch/1' or degrees/1' or rotations/1'
_CFGVMAXI	maximum interpolation speed	f	m/1' or inch/1' or degrees/1' or rotations/1'
_CFGPHINV	encoder phase inversion	b	0=no inversion, 1=inversion
_CFGRFINV	reference inversion	b	0=no inversion, 1=inversion
_CFGZIND	enable on-index position reset	b	0=disabled, 1=enabled
_CFGTRPP	type of acceleration/ deceleration ramp in point-to-point mode	b	0=linear, 1= 'S' shaped , 2= double 'S' shaped
_CFGKFFA	acceleration feed forward	f	
_CFGKFFAI	interpolation acceleration feed forward	f	
_CFGSRPP	step by step ramp start speed	f	m/1' o inch/1' o gradi/1' o giri/1'
_CFGACC	acceleration time from 0 to _CFGVMAX	i	msec
_CFGDEC	deceleration time from _CFGVMAX	i	msec
_CFGACCI	acceleration time from 0 to _CFGVMAXI	i	msec
_CFGDECI	deceleration time from _CFGVMAX to 0	i	msec
_CFGQLP	positive axis limit	d	position
_CFGQLN	negative axis limit	d	position
_CFGKP	proportional coefficient	f	
_CFGKI	integral coefficient	f	
_CFGKD	derivative coefficient	f	
_CFGKFF	feed forward	f	percentage
_CFGKPS	slave axis proportional coefficient	f	
_CFGKIS	slave axis integral coefficient	f	
_CFGKDS	slave axis derivative coefficient	f	
_CFGQEAP	positive loop error	d	position
_CFGQEAN	negative loop error	d	position
_CFGKPI	interpolation proportional coefficient	f	
_CFGKII	interpolation integral coefficient	f	
_CFGKDI	interpolation derivative coefficient	f	
_CFGTMINP	minimum positive voltage	f	volt
_CFGTMINN	minimum negative voltage	f	volt
_CFGSTMINP	positive threshold voltage	f	volt
_CFGSTMINN	negative threshold voltage	f	volt
_CFGESC	axis moving timeout	i	msec

costant	description	type	return value
_CFGDSE	enable dynamic servoerror	b	0=disabled, 1=enabled
_CFGAEN	enable automatic adjust	b	0=disabled, 1=enabled
_CFGOFFSET	adjust voltage - initial offset	f	volt
_CFGCEE	incorrect encoder connection position	d	position
_CFGNOTCH	notch filter frequency	i	Hz
_CFGBUFI	integrative calculation dimension buffer	i	[1, 200]
_CFGQAP	positive quiescent threshold	d	
_CFGQAN	negative quiescent threshold	d	
_CFGTRI	type of acceleration/ deceleration ramp in interpolation mode	f	0=linear, 1= 'S' shaped ,2= double ‘S’ shaped
_CFGKFFI	interpolation feed forward	f	percentage
_CFGAAF	wait axis still	b	0=disabled, 1=enabled
_CFGENCTYPE	type of encoder	i	0=simulated or absent, 1=real
_SRPP	step by step ramp start speed	f	m/1' or inch/1' or degrees/1' or rotations/1'
_ACC	acceleration time from 0 to _VMAX	i	msec
_DEC	deceleration time from _VMAX to 0	i	msec
_ACCI	acceleration time from 0 to _VMAXIin interpolation mode	i	msec
_DECI	deceleration time from_VMAXI to 0 in interpolation mode	i	msec
_QLP	positive axis limit	d	position
_QLN	negative axis limit	d	position
_KP	proportional coefficient	f	
_KI	integral coefficient	f	
_KD	derivative coefficient	f	
_KFF	feed forward	f	percentage
_KPS	slave axis proportional coefficient	f	
_KIS	slave axis integral coefficient	f	
_KDS	slave axis derivative coefficient	f	
_QEAP	positive loop error	d	position
_QEAN	negative loop error	d	position
_VEL	point- to-point speed	f	m/1' or inch/1' or degrees/1' or rotations/1'
_VELI	interpolation speed	f	m/1' or inch/1' or degrees/1' or rotations/1'
_MODE	axis functioning mode	b	1=normal, 2=free, 8=interpol., 10=coord.
_PHINV	encoder phase inversion	b	0=no inversion, 1=inversion
_RFINV	reference inversion	b	0=no inversion, 1=inversion
_ZIND	enable on-index position reset	b	0=disabled, 1=enabled
_KPI	interpolation proportional coefficient	f	
_KII	interpolation integral coefficient	f	

costant	description	type	return value
_KDI	interpolation derivative coefficient	f	
_KFFI	interpolation feed forward	f	percentage
_KFFA	acceleration feed forward	f	percentage
_KFFAI	interpolation acceleration feed forward	f	percentage
_ESC	axis moving timeout	i	msec
_CEE	incorrect encoder connection position	d	position
_NOTCH	notch filter frequency	i	Hz
_BUFI	integrative calculation dimension buffer	i	[1,200]
_QAP	positive quiescent threshold	d	
_QAN	negative quiescent threshold	d	
_QEAPINV	positive loop error limit in inversion	d	
_QEANINV	negative loop error limit in inversion	d	
_OFSCOORD	offset position coordinated move	d	
_MS	axis typology master or slave	b	0=not in chain,4=master,5=slave
_QENC	encoder position	d	position
_QR	real position	d	position
_RIS	resolution used by the axis	d	
_ST	axis state	b	1=accel., 2=regime, 3=decel.,4=position, 5=wait big win.,6=wait axis stop,7=wait small win, 8=start
_QT	theoretical position	d	position
_EA	loop error	d	position
_FF	feed forward	i	
_VC	current speed	f	
_P	proportional correction	i	
_I	integral correction	i	
_D	derivative correction	i	
_FLGS	axis flags	b	
_VCR	real speed	f	
_ADJUST	axis compensation offset	i	whole number showing the tension to be transmitted to the drive, as seen from the side of the DAC axis card. The full scale of the drive is 10 Volt and that of the DAC is 32767.
_DAC	DAC value	i	whole number representing the tension to be transmitted to the drive, as seen from the side of the DAC axis card. The full scale of the drive is 10 Volt and that of the DAC is 32767.
_ACCINST	instantaneous acceleration value	f	
_FFA	acceleration feed forward	i	

costant	description	type	return value
_GONETIME	elapsed time from the beginning of the movement	f	sec (0 for slave axis and step-by-step axis)
_RESTIME	time left until the end of the movement. The values are related to the allocated movement in the buffer when requested.	f	sec (0 for slave axis, coordinated moving axes and step-by step axis)
_GONESPACE	space from the beginning of the movement. The values are related to the allocated movement in the buffer when requested.	f	percentage (100 for slave axis, interpolated moving axes and step-by-step axis)
_RESSPACE	space left until the end of the movement. The values are related to the allocated movement in the buffer when requested.	f	percentage (100 for slave axis, interpolated moving axes and step-by-step axis)
_AXESJERK	enabling the jerk control on the axis	b	1=enabled control, 0=control not enabled
_MOVEJERK	enabling the jerk control on the movement on which the axis is engaged	b	1=enabled control, 0=control not enabled
_MOVETYPE	axis motion on which the axis is engaged.	b	1=classical interpolated movement, 2=interpolated multi-axis motion, 3= coordinated movement 4= movement point-to-point, 5=movement in chain (slave axes only)
_PARTYPESET	type of parameter axes in use during the movement	i	1=interpolation, 0=point-to-point
_AXINRIFLOC	current axe in a local reference system	i	1=yes, 0=no
_QTARGETTOOL	target position of the axis. In case of ISO interpolation target position of the coordinate of the tool point of the axis	d	
_QREALTOOL	real position of the axis. In case of ISO interpolation real position of the coordinate of the tool point of the axis	d	
_BACKLASH	value of the mechanical clearance defined for the axis	d	
_DISABLED	disabling an axis	b	1=disabled axis, 0= enabled axis
_DYNLIMIT	enabling dynamic numeric control of axis limits	b	1=enabled control, 0= disabled control
_AXESFEED	override feedrate value currently applied to the axis	f	
_CORRLIN	kind of linearity correction in use	i	0=no correction in use, 1=self correction, 2=crossed correction,

costant	description	type	return value
			3=self correction with crossed correction
_VELISO	the tool point speed during the ISO - movement	f	
_ISOSTOPS	number of the forced stops of the interpolated movement due to borderline situations of the lookahead	i	
_CURRATIO	value of the chaining ratio currently used	d	
_DYNRATIO	returns, if a dynamic change of the chaining ration is in execution	i	0=no, 1 = yes
_RESBLOCK	number of total queued displacement blocs in the movement (current value)	i	
_EXECBLOCK	number of the displacement blocs still to be performed	i	
_TOTALBLOCK	number of performed displacement blocs	i	
_SWITCHENC	monitors if the encoders are being exchanged	i	-1=the axis does not use the SWITCHENC instruction, 0= a SWITCHENC instruction has been executed, but the axis is using its encoder, 1= a SWITCHENC instruction has been executed and the axis is using the encoder of the counting axis
_QOFSENC	encoder offset value	d	

Point-to-point Movement

SETACC

Syntax

SETACC	**axis, [value]**

Arguments

axis	name of axis device
value	constant or variable. Acceleration time

Description

It assigns to the **axis** the acceleration time indicated by **value**. Acceleration time is expressed in milliseconds.
If **value** is omitted, it assigns the configuration parameter. If the instruction is placed between two instructions MOVABS or MOVINC, the first movement instruction (with stop of the movement)is executed, using the acceleration and deceleration parameters previously set. The second instruction is executed, when the new parameter of acceleration are applied. SETACC instruction has an effect only on the movements coming after its execution.
If the specified **value** is smaller than the configuration parameter then the latter is taken.

See also SETDEC, SETACCI and SETDECI.

SETDEC

Syntax

SETDEC	**axis, [value]**

Arguments

axis	name of axis device
value	constant or variable. Deceleration time

Description

It assigns to the **axis** the deceleration time indicated by **value**. Deceleration time is expressed in milliseconds.
If **value** is omitted, the configuration parameter is taken. If the instruction is placed between two instruction, MOVABS or MOVINC, the first movement instruction (with stop of the movement), is executed using the acceleration and deceleration parameters previously set. The second instruction is executed, when the new parameter of deceleration are applied. SETACC instruction has an effect only on the movements coming after its execution.
If the specified **value** is smaller than the configuration parameter then the latter is taken.

See also SETACC, SETACCI and SETDECI.

SETDERIV

Syntax

SETDERIV	**axis [, value]**

Arguments

axis	name of axis device
value	constant or variable. Char and integer variables are not allowed

Description

It assigns the **value** *derivative action coefficient* to the **axis**.
If **value** is omitted, the configuration derivative action coefficient is used.
The instruction can not be applied to a step-by-step motor.
See also instruction SETDERIVI.

SETFEED

Syntax

SETFEED	**axis, value**

Arguments

axis	name of axis device
value	constant or variable. It represents the feed rate override percentage

Description

It modifies the percentual **value** of the **axis** feed rate override in relation to *point-to-point movements*. See also SETFEEDI.

SETFEEDF

Syntax

SETFEEDF	**axis [, value]**

Arguments

axis	name of axis device
value	constant or variable. Feed rate override percentage

Description

It assigns the **value** *feed forward percentage* to the **axis**.
If **value** is omitted, the configuration feed forward coefficient is used.
If the instruction is applied to a step-by-step motor a system error is generated. The same happens if the **value** variable is set on a value which is not included between 0 and 100.
See also instructions SETFEEDFI, SETFEEDFA, SETFEEDFAI.

Note

If the instruction is applied to an engine with SLM command and plate speed higher than 3750 RPM, the maximum feed forward adjustable value is 50. This is because the engine with Commands SLM performs a scaling on internal reference speed sent by the control.

SETFEEDFA

Syntax

SETFEEDFA	**axis [, value]**

Arguments

axis	name of axis device
value	constant or variable. Feed forward percentage

Description

It assigns to the **axis** the acceleration *feed forward percentage* **value** for point-to-point movements.
If **value** is omitted, the configuration feed forward coefficient is used.
If the instruction is applied to a step-by-step motor a system error is generated. The same happens if the **value** variable is set on a value which is not included between 0 and 100.
See also instructions SETFEEDF,SETFEEDFI,SETFEEDFAI.

Note

If the instruction is applied to an engine with SLM command and plate speed higher than 3750 RPM, the maximum feed forward adjustable value is 50. This is because the engine with Commands SLM performs a scaling on internal reference speed sent by the control.

SETINTEG

Syntax

SETINTEG	**axis [, value]**

Arguments

axis	name of axis device
value	constant or variable. Integral action coefficient. Char and integer variables are not allowed.

Description

It assigns the **value** *integral action coefficient* to the **axis**.
If **value** is omitted, the configuration integral action coefficient is used.
The instruction can not be applied to step-by-step motors.
See also instruction SETINTEGI.

SETMULTIFEED

Syntax

SETMULTIFEED	**axis1, value1, axis2, value2 [, axis3, value3 [, ..., axis16, value 16]]]**

Arguments

axis1...axis16	name of devices of type axis
value1...[...value16]	constant or variable. It represents the feed rate override percentage

Description

It modifies the **feed rate** override percentage value of the indicated **axes** indicated as far as the *point- to-point movements* are concerned. For each axis a different value can be set.

SETPROP

Syntax

SETPROP	**axis [, value]**

Arguments

axis	name of axis device
value	constant or variable. Proportional action coefficient. Chars and integers are not allowed

Description

It assigns the *proportional action coefficient* **value** to the **axis**.
If **value** is omitted, the configuration proportional action coefficient is used.
The instruction can not be applied to step-by-step motors.
See also instruction SETPROPI.

SETVEL

Syntax

SETVEL	**axis [, speed]**

Arguments

axis	name of axis device
speed	float constant or float variable

Description

It sets the highest **speed** of the axis for point-to-point movements.
Speed is expressed in the axis measuring unit, specified in configuration.
If the programmed **value** is higher than the value of configuration, the latter is used.
It the **speed** argument is omitted, the configuration value is used. Only positive **speed** values are allowed.
See instruction SETVELI.

Example

Axis Homing routine

Interpolated Movement

LOOKAHEAD

Syntax

LOOKAHEAD	**[value]**

Arguments

value	constant or variable. Look ahead value

Description

Sets the interpolator look ahead value. Look ahead is the number of interpolation blocks that will be processed before starting axes motion. It allows generation of optimized speed profiles, specifically when using "S" shaped acceleration and deceleration ramps.
In case **value** parameter is not specified, a default look ahead of 512 blocks is assumed.
Maximum allowed value is **4096/channelsnumber** where **channelsnumber** is the number of interpolation channels as defined in module configuration. Minimum allowed value is 256.

NOTICE: an interpolation block is constituted by the set of information associated to any instruction of interpolated displacement (e.g. LINEARABS).

Example

```
LOOKAHEAD 1024
```

SETACCI

Syntax

SETACCI **axis1 [, …, axis6] [, value]**

Arguments

axis1,[…axis6]	name of axis device
value	constant or variable. Acceleration time

Description

It assigns to axes **axis1** and **axis2** the interpolation movement acceleration time indicated by **value**. Time is expressed in milliseconds. If **value** is omitted, the configuration parameter is taken instead.

See also SETACC, SETDEC and SETDECI.

SETACCLIMIT

Syntax

SETACCLIMIT **axis,[value]**

Arguments

axis	name of axis device
value	operating time constant

Description

It enables and disables the automatic calculation of interpolation regime speed according to the acceleration tolerated by the axes. The **value** parameter is a time constant used to define the speed limit tolerated by the **axis**, in milliseconds. This parameter is optional. If omitted, the macro will disable the automatic calculation. A standard value for this parameter is 30 milliseconds. If this time is further reduced, the profile will slow down making movement more gentle. By increasing this time, the opposite effect is obtained. This instruction can't be applied to helical interpolations.

SETACCSTRATEGY

Syntax

SETACCSTRATEGY **axis, [value]**

Arguments

axis	name of axis device
[value]	acceleration strategy

Description

Allows the selection of the type of acceleration wanted for the following interpolation movements. The instruction is executed for all the axes involved in the interpolation.
There are two admissible values for the parameter **value:** 0 and 1. If the value 0 is passed, the usual acceleration strategy is adopted (the least of the axes involved in the interpolation is chosen as profile acceleration). If the value is equal to 1 the highest acceleration that the individual axes can support is taken (considering the individual components). In this latter case, only the linear interpolation strokes will be considered and the algorithm will work only so long as the acceleration and deceleration ramps are contained in the same interpolation stroke.

SETAXPARTYPE

Syntax

SETAXPARTYPE **axis, [value]**

Arguments

axis	name of the axis device
[value]	variable or integer constant.

Description

When a multilinear interpolation is performed, this instruction allows you to change the axis parameter set in use, changing from the typical parameters of the interpolation (**value** =1) to those used for the

point-to-point movement (**value** = 0). If the variable **value** is omitted, the resolution value used is the interpolation one.
The parameter set change can only be made if the axis is still in POSITION state, otherwise the instruction generates the system error no. 4101 "Inconsistent axis *AxisName* management".

SETCONTORNATURE

Syntax

SETCONTORNATURE	**[value1[,value2]]**

Arguments

value1	constant or variable. Maximum contouring angle
value2	constant or variable. Maximum slowdown angle

Description

Sets the minimum angle between the tangents of two trajectories carried out in interpolation. If the angle is exceeded, the machine will not carry out the contouring, that is, the axes will stop at the end of the first trajectory and then restart along the second one. For this reason a *maximum contouring angle* is defined as **value1** and represents the maximum angle between two displacement lines, below which the movement does not stop. If the angle between two displacement blocks is greater than the maximum contouring angle, the movement stops. To avoid the stop, a maximum deceleration angle (**value2)** can be set. If the angle between two displacement blocks is included between the*maximum contouring angle and the maximum deceleration angle,* the movement does not stop, but only slows down. Thus, the *maximum deceleration angle* represents the angle over which the movement must be compulsorily stopped. For angles less than the maximum contouring angle the movement **does not slow down**, for angles between the maximum contouring angle and the maximum deceleration angle the movement **stops.**
Value1 and **value2** are optional parameters; if both are not set, 15 degrees are taken on as a default value. If only the first parameter is set, *maximum deceleration angle* is equal to *the maximum contouring angle.* Deceleration feature is disabled when the *maximum deceleration angle* is less or equal to *the maximum contouring angle.* The maximum deceleration *angle* is equal to 180 degrees. If an greater value is set, the generates the following error no. 4399 "Parameter out of range".
Deceleration feature is enabled only if the instruction JERKSMOOTH is active; however, the contouring is always active.

Nota

L'uso di questa istruzione è associato all'uso delle istruzioni JERKSMOOTH e SETSLOWPARAM e it is only applied in the movements with classic interpolation (LINEARABS, LINEARINC, CIRCABS, CIRCINC, HELICABS, HELICINC instructions).

SETDECI

Syntax

SETDECI	**axis1 [, ..., axis6] [, value]**

Arguments

axis1,[...axis6]	name of axis device
value	constant or variable. Deceleration time

Description

It assigns to axes **axis1** and **axis2** the interpolation movement deceleration time indicated by **value**. Time is expressed in milliseconds. If **value** is omitted, the configuration parameter is taken instead.

See also SETACC, SETDEC, and SETACCI.

SETDERIVI

Syntax

SETDERIVI	**axis [, value]**

Arguments

axis	name of axis device
value	constant or variable. Derivative action coefficient. Char and integer variables are not allowed

Description

It assigns to the **axis** the **value** *derivative action coefficient* during axis interpolation movement.
If **value** is omitted, the configuration derivative action coefficient is used.
The instruction can not be applied to a step-by-step motor.
See also instruction SETDERIV.

SETFEEDFAI

Syntax

SETFEEDFAI	**axis [, value]**

Arguments

axis	name of axis device
value	constant or variable. Feed forward percentage

Description

It assigns to the **axis** the acceleration *feed forward percentage* **value** for interpolation movements
If **value** is omitted, the configuration feed forward coefficient is used.
If the instruction is applied to a step-by-step motor a system error is generated. The same happens if the **value** variable is set on a value which is not included between 0 and 100.
See also instructions SETFEEDF,SETFEEDFI,SETFEEDFA.

Note

If the instruction is applied to an engine with SLM command and plate speed higher than 3750 RPM, the maximum feed forward adjustable value is 50. This is because the engine with Commands SLM performs a scaling on internal reference speed sent by the control.

SETFEEDI

Syntax

SETFEEDI	**axis, value**

Arguments

axis	name of axis device
value	constant or variable. It represents the feed rate override percentage

Description

It modifies the percentual **value** of **axis** feed rate override in relation to interpolation movements.
See also instruction SETFEED.

SETFEEDFI

Syntax

SETFEEDFI	**axis [, value]**

Arguments

axis	name of axis device
value	constant or variable. Feed forward percentage

Description

It assigns to the **axis** the feed forward percentage **value** for interpolation movements.
If the argument **value** is omitted, the system takes the feed forward percentage set in the configuration parameters of the concerned axis device.
The instruction can not be applied to step-by-step motors.
The **value** variable admits values included between 0 and 100.
See also instructions SETFEEDF,SETFEEDFA,SETFEEDFAI.

Note

If the instruction is applied to an engine with SLM command and plate speed higher than 3750 RPM, the maximum feed forward adjustable value is 50. This is because the engine with Commands SLM performs a scaling on internal reference speed sent by the control.

SETINTEGI

Syntax

SETINTEGI **axis [, value]**

Arguments

axis	name of axis device
value	constant or variable. Integral action coefficient. Char and integer variables are not allowed.

Description

It assigns to the **axis** the *integral action coefficient* **value** used during axis interpolation movements.

If **value** is omitted, the configuration integral action coefficient is used.
The instruction can not be applied to step-by-step motors.
See also instruction SETINTEG.

SETPROPI

Syntax

SETPROPI **axis [, value]**

Arguments

axis	name of axis device
value	constant or variable. Proportional action coefficient. Chars and integers are not allowed

Description

It assigns to the **axis** the *proportional action coefficient* **value** used during axis interpolation movements.
If **value** is omitted, the configuration proportional action coefficient is used.
The instruction can not be applied to step-by-step motors.
See also instruction SETPROP.

SETSLOWPARAM

Syntax

SETSLOWPARAM **axis [,value1,value2]**

Arguments

axis	name of device of axis type
value1	constant or variable double. General reduction factor
value2	constant or variable double. Inversion reduction factor

Description

This instruction modifies the parameters needed to calculate the deceleration, where deceleration features are active while contouring (see instruction SETCONTORNATURE).
Deceleration speed is initially calculated for each axis in a technical way. In case of motion reversal, it can be reduced using **value2.** Later, among all the calculated speed rates, the minimum speed rate is taken into account, in order to comply with the dynamic of the more limiting axis. Finally, a further reduction of the deceleration speed of a factor which depends on **value1,** is possible.

If **value1** or **value2** are omitted, values by default are taken on, so that both the parameters do not take effect. The **value1** parameter represents the reduction percentage value of the theoretical speed slowdown. The applied slowdown speed is equal to (100 **value1**((100-valore1)/100))* theoretical speed. Maximum reduction value is equal to 100. In this case the resulting speed corresponds to 1% of the theoretical speed. Vice versa, when the value is 0 or it is omitted, the default value, that is, the entire theoretical speed, is taken into account.
The parameter **value2** represents the percentage of reduction, between 1 and 10 times, of the theoretic slowdown, should an axis reverse its motion. In particular, when **value2** is 100, the speed rate drops by 10 times. Vice versa, when it is equal to zero or it is omitted, the speed rate does not drop.

The instruction generates the system error 4399 "Parameter out of range", when the value set is less than 0 or greater than 100. It is important to remember that if the parameter **value1** is omitted, also the **value2** parameter must be omitted.

Note

This instruction requires the instructions JERKSMOOTH and SETCONTORNATURE and is only effective with classical interpolation (instructions LINEARABS, LINEARINC, CIRCABS, CIRCINC, HELICABS, HELICINC).

SETVELI

Syntax

SETVELI	**axis1 [, ..., axis6] [, speed]**

Arguments

axis1 [...axis6]	name of axis device to be interpolated
speed	float constant or float variable

Description

It sets the highest **speed** of **axis1** and **axis2**, for interpolation movements.
Speed is expressed in the axis measuring unit, specified in the configuration parameter. If the **speed** argument is omitted, maximum configuration speed is taken.
Step-by-step axes can be used in this instruction only if they are controlled by a TRS-AX remote.
See instruction SETVEL.

SETVELILIMIT

Syntax

SETVELILIMIT	**axis, speed**

Arguments

axis	name of axis device
speed	float constant or float variable

Description

It sets the single **speed** components of the **indicated axis,** for interpolated movements.
The speed is expressed in the UOM of the axis.

Coordinated Movement

SETFEEDCOORD

Syntax

SETFEEDCOORD	**axis, value1, value2**

Arguments

axis	name of the device of axis type
value1	constant or variable double. It represents the maximal percentage of feed rate override.
value2	constant or variable integer. It represents the number of real time where the feed rates variation has to be applied.

Description

This modifies **value1** percentage of the **axis** feed rate's maximal instantaneous variation. Feed rate is not changed anymore in the time, expressed as a real time and defined into the **value 2** variable. In other words, after applying a variation of feedrate override of **value1**, as highest value, by a Real Times number of **value2,** any new feedrate variation cannot be applied. The combination of these two parameters defines a sort of acceleration/deceleration, that the axis can sustain. By modulating these two parameters, we can obtain some "step ramps" of the ramp required.

Note

For each axis involved in the coordinate move feedrate value and time should be set, otherwise the default values **value 1**=100 and **value2**=1 are taken. During the execution of the coordinated move (instruction COORDIN), the system calculates again the parameters **value1** and **value2** to apply to

the move according to all the involved axes' parameters. The motionless axes are excluded from the control. Both parameters are calculated as follows:
value1: minumum value set on the moving axis;
value2: value obtained dividing value1 by the lowest ratio **value1/value2.**

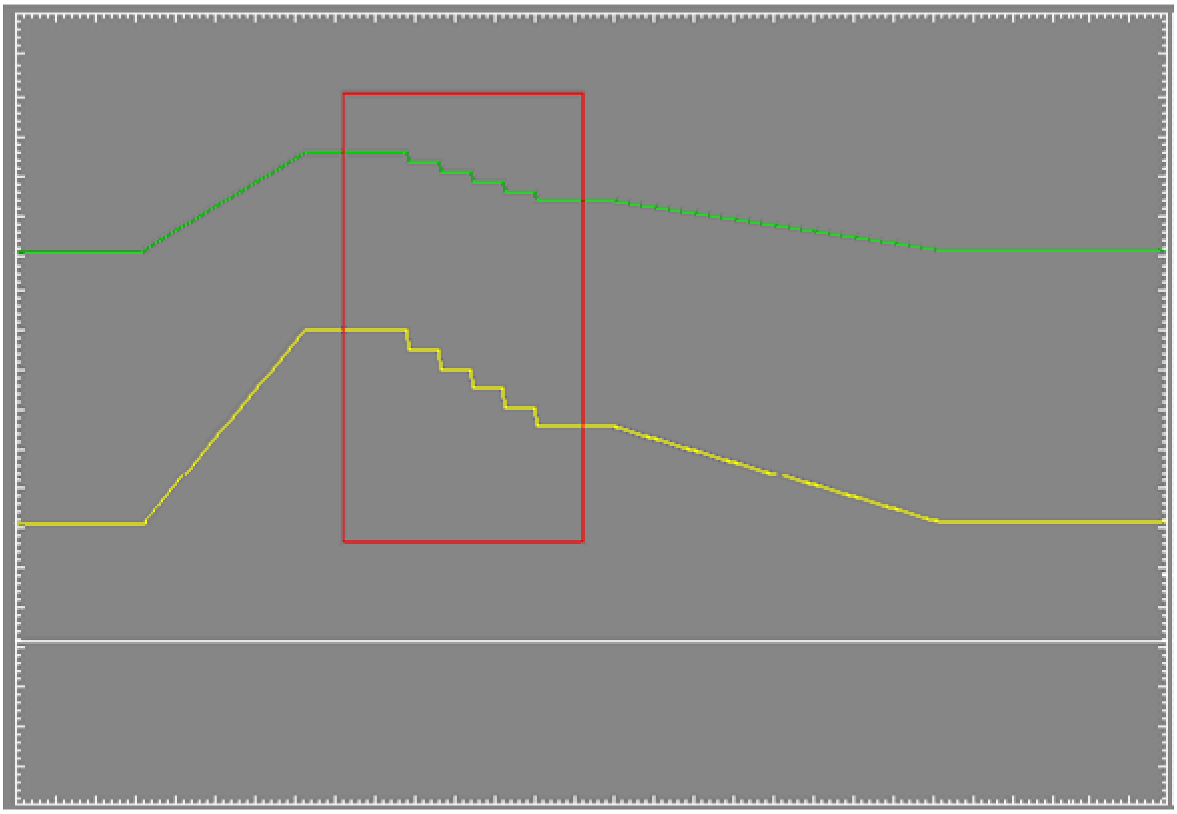
Exemple

```
;
Function CoordinatedMove

      Setquote    X,0
      Setquote    Y,0
      Setquote    Z,0

      setFeedCoord           X, 20, 80
      setFeedCoord           Y, 10, 1
      setFeedCoord           Z, 3, 3

      coordin     matrix, deltaT, UP, rigaInit, rigaEnd,mask, _
                  X,columnX, Y,columnY, Z,columnZ
      waitstill   x,y,z

fret
```

Suppose that in a specific passage of the coordinated move the z-axis does not move. Set parameters result to be

Max_Variation = **10**
Delta_T = 10 / 0.25 = **40**

Therefore we have to following trace of oscilloscope, where the speed rate profile of the X-axis is marked in green and that of the Y-axis is marked in yellow.



SETOFFSET

Syntax

SETOFFSET	**axis, position**

Arguments

axis	name of axis device
position	constant or variable. Offset for coordinated movements

Description

It allows you to apply an offset to the position of a coordinated movement.
The offset specified by the position parameter will be used in later coordinated movements, adding the indicated position to all the positions in the table.
See also instruction COORDIN.

Chained Movement

RATIO

Sintassi

RATIO	**axis, [value]**

Argomenti

axis	name of axis device
value	costant or variable. Reduction ratio.

Descrizione

Sets the chaining ratio of a slave axis with respect to its master. Slave axis movements will be scaled with respect to master movements by the set chaining ratio. If the **value** parameter is omitted, the ratio is reset to 1.0 (identical movements). Instruction generates system error if executed when the axis is not in slave state and the corresponding master axis is not in position state.
See CHAIN instruction.

Esempio

```
CHAIN          X, Y
RATIO          Y, 0.5      ; reduction ratio 1/2

MOVABS         X, 100      ; Y axis will move to position 50
WAITSTILL      X
```

SETDYNRATIO

Syntax

SETDYNRATIO	**axis, value**

Arguments

axis	name of the axis device
value	constant or double variable

Description

This instruction allows the chaining ratio to be changed in a dynamic way during the movement of the master axis. It is possible to apply the new value of the chaining ratio, even though the previous variation has not ended. The declared **axis** must be a slave axis.
If the instruction is executed with master axis at the state POSITION, the new value of the chaining ratio **value** is instantaneously applied.
The variation of the chaining ratio occurs by means of a linerar acceleration (or deceleration) ramp. The acceleration value employed is given by the acceleration of the Master-axis currently used for the point-to-point movement. This means that it is also possible to modify this ramp by setting a new acceleration value using the instruction SETACC.
This instruction can generate following system error:

- "4101: Inconsistent axis AxisName management", in the event that the **axis** declared is not a slave axis.

Generic Parameters

DYNLIMIT

Syntax

DYNLIMIT **axis, state**

Arguments

axis	name of axis device
state	predefined constant Permitted values: **ON** enabling dynamic controls of the axis limits **OFF** disabling dynamic controls of the axis limits

Description

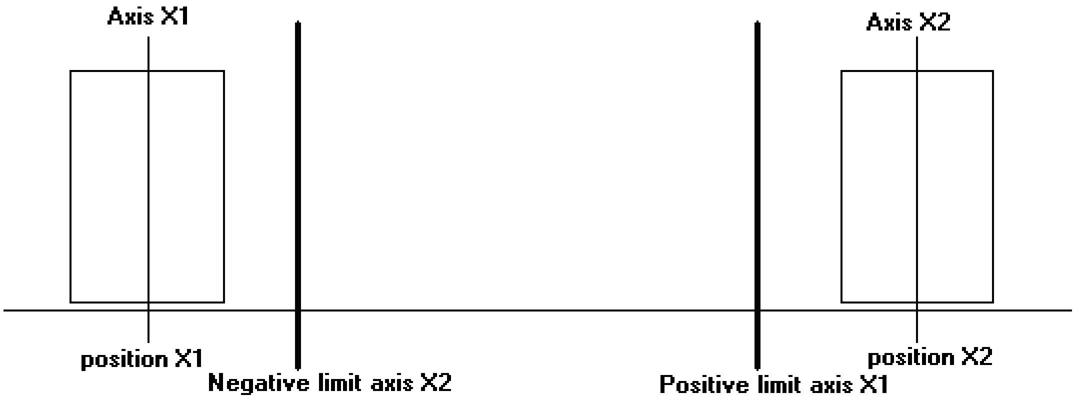
It enables or disables the dynamic test of exceeded axis limit.
What distinguishes the dynamic test of exceeded axis limit from the static test of exceeded axis limit is that the first one verifies at each real time that the axis exceeds its limits, according to its current speed rate and to its maximum deceleration. The test of static type, instead, verifies instant by instant that the current arrival position of each axis is located within the positive or negative set axis limits. Furthermore, before the beginning of the move, the test of static type verifies if the positions given by the movement instructions exceed the set limits.
Before a DYNLIMIT instruction SETLIMPOS and SETLINMNEG instructions must be set, in order to define the new limits.

Example

Check of the axes limits according to both typologies of static and dynamic test, with axes on the same movement directrix.

Static test.
In a generic movement the **Axis X1** cannot exceed the initial positive limit given by the **Axis X2** position. Axes limit check generates a system error no. 4108 " Axis X1: final position exceeding the software limit".

Dynamic test
It verifies in a generic movement that the instantaneous **X1 position** is located, with a proper sign and according to the movement direction of the axis, within the axis limits decreased of the minimum stop space of the same axis. The minimum stop space is calculated according to the instantaneous speed rate and to the deceleration set into the configuration of the point-to-point movement. Furthermore, this test does not verify before the beginning of the movement, if the positions given by the movement instructions exceed the set limits.



ENABLESTARTCONTROL

Syntax

ENABLESTARTCONTROL **axis, [timeout]**

Arguments

axis	name of axis device
timeout	variable. Wait timeout

Description

This instruction allows for **timeout** to be enabled and selected to control the non-start up or sudden stop of the axis.
If the axis does not move by at least 2 steps in 200 RealTime when movement is executed, a system error is generated.
If the **timeout** parameter is set to zero, the control is disabled. The instruction is not enabled if the theoretical speed is slower than two steps in 200 RealTime or if the movement ends in less than 200 RealTime.

Example

```
; axes starting timeout equal to 10 ms
ENABLESTARTCONTROL    x, 10
```

NOTCHFILTER

Sintassi

NOTCHFILTER	**axis, [value]**

Argomenti

axis	name of axis device
value	constant or variable. Frequency value [Hz]. Valid values are in the range **0** to **500**.

Descrizione

Sets the notch filter's cut-off frequency for the axis specified. If **value** equals 0, the filter is disabled.
If the **value** parameter is omitted, the value set in configuration will be used.

Esempio

```
; frequency cut-off 97 Hz
NOTCHFILTER              X, 97
```

RESLIMNEG

Syntax

RESLIMNEG	**axis**

Arguments

axis	name of axis device

Description

It disables the test on the negative limit of the indicated **axis**.
These instructions are usually used in homing routines to search for home switches, allowing the axes to exceed the set configuration values.
See also instructions SETLIMNEG, SETLIMPOS, RESLIMPOS.

Example

Axis Homing routine

RESLIMPOS

Syntax

RESLIMPOS	**axis**

Arguments

axis	name of axis device

Description

It disables the test on the positive limit of the indicated **axis**.

These instructions are usually used in homing routines to search for home switches, allowing the axes to exceed the set configuration values.
See also instructions RESLIMNEG, SETLIMPOS, SETLIMNEG.

Example
Axis Homing routine

SETADJUST

Syntax

SETADJUST **axis, state [value]**

Arguments

axis	name of axis device
state	predefined constant. Possible values are: **ON** to enable **OFF** to disable
[value]	float variable or constant. Voltage [Volt]

Description

It enables or disables, on the specified **axis**, the automatic calculation of offset recovery, that is the ADJUST.
The adjust allows you to compensate slight position offsets at the end of axis movement. It is normally enabled.
It can be convenient to disable the adjust for axes moved by motors with a high position hysteresis which would not benefit by using this control function.
When the adjust is reactivated after having been disabled, the control does not consider the value calculated previously, so the instruction can also be used to delete the accumulated adjust of an axis without having to restart the control.
When the third parameter is present, offset is set on the indicated **value** apart from automatic ADJUST activating or deactivating. The use of this instruction allows you to compensate via software a speed reference offset instead of compensating it on drive, even if the compensation on drive is to be preferred.
The instruction can only be used with analog controlled axes (AlbNT cards).

SETBACKLASH

Syntax

SETBACKLASH **axis, value**

Arguments

axis	name of the device type axis
value	variable or float constant. Backlash value.

Description

This instruction allows you to reduce or eliminate the effects of mechanical slackness on the **axis** trajectory. The **value** of the game that can be set should be between 0.0 and 3.0. This value is independent of the unit of measure choice. Special situations occur in the following cases:

- if the axis is disabled, backlash recovery function is not applied, even if requested.
- In case of vertical axis, given the particular configuration, it does not occur any backlash.
- In case of axis with a load of great inertia, there may be a partial or at times a total load compensation. As a matter of fact, due to the mass of the load, the motion of the axes could stop later than the engine. The resulting positioning of the reduction gear teeth as regards the teeth positioning of the driving gear can reduce or even cancel the backlash.
- Visualization of the real quotes and encoder of the axis, sampled by the oscilloscope on the points, where the backlash recovery is activated (movement reversal), shows a pick equivalent to the backlash value itself.
-

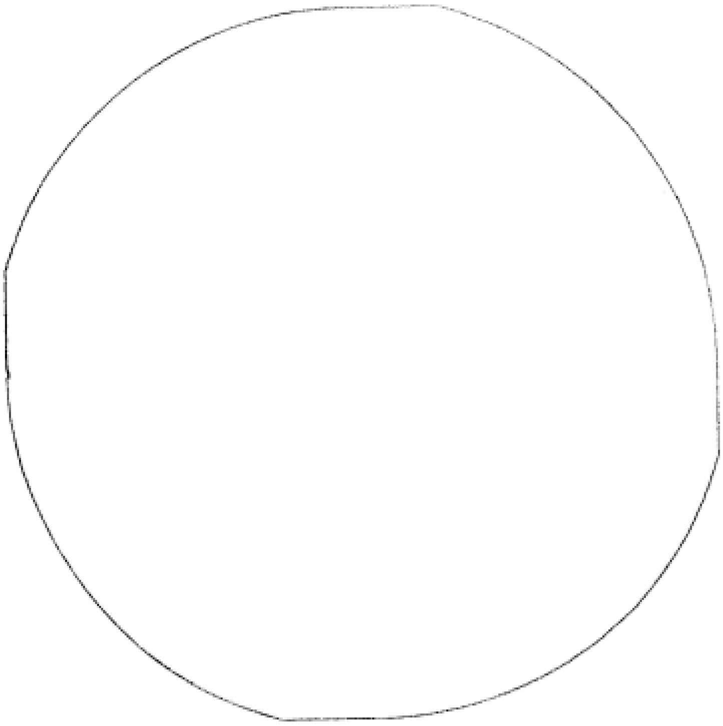
The instruction generates a system error, in case of use:
- on step-to-step, not controlled by TRS-AX remotes, counting, virtual axes
- on step-to-step axes, controlled by TRS-AX remotes with simulated encoder

Exemple

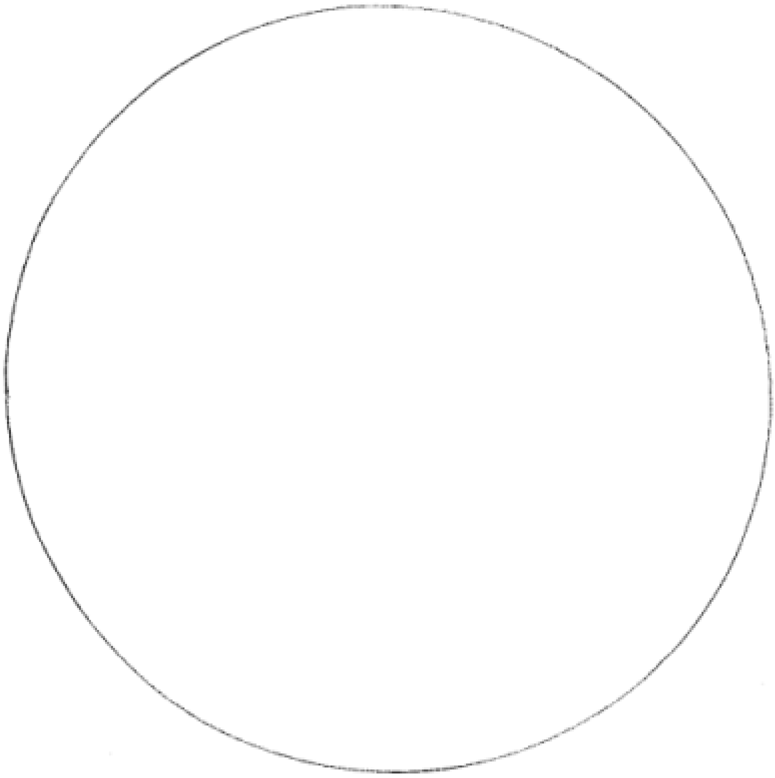
```
 ; Function whose backlash recovery is disabled  (red line in the drawing)
  SETQUOTE        X, 0
```

```
SETQUOTE        y, 0
SETVELI         X, 1.0
CIRCLE          X,Y,cw,100,90
WAITSTILL       X,Y

; Function whose backlash recovery is enabled
; (black line in the drawing)
SETQUOTE        X, 0
SETQUOTE        y, 0
SETVELI         X, 1.0
SETBACKLASH     X, 1.9
SETBACKLASH     y, 1.8
CIRCLE          X,Y,cw,100,90
WAITSTILL       X,Y
```

Carrying out the two functions generates two different traces.
The first figure shows the interpolation on two axes, that present a backlash in the mated engine-reduction gear.



The second figure represents the same interpolation, but containing the instruction of backlash recovery.



SETBIGWINFACTOR

Syntax

SETBIGWINFACTOR **axis, value**

Arguments

axis	name of axis device
value	double constant or variable. Multiplication factor for the calculation of the big window

Description

This instruction allows you to modify the multiplication factor for the calculation of the big window on **the axis** requested. To calculate the big window, we need to multiply the variable **value** by the parameter defined in the axes configuration of the position arrival window. The **value** that can be set should be included between 1 and 257 first and final value excluded. Default value is 4.0.

SETDEADBAND

Syntax

SETDEADBAND **Asse,VMinPos,VMinNeg,VThrePos,VThreNeg**

Arguments

axis	name of axis device
VMinPos	float variable or constant. Minimum positive voltage [Volt]
VMinNeg	float variable or constant. Minimum negative voltage [Volt]
VThrePos	float variable or constant. Positive threshold [Volt]
VThreNeg	float variable or constant. Negative threshold [Volt]

Description

It sets the minimum voltage for the indicated axis. The minimum (positive/negative) voltage values are added to the theoretical reference voltage (positive/negative) , if this exceeds the (positive/ negative) threshold value selected. If the theoretical reference voltage falls within threshold values, the actual reference voltage is forced to zero. Minimum voltage management can be disabled, setting all values to zero. The threshold values must always be below or equal to relative minimum voltage values.

When the system starts up, minimum voltage management is disabled.

SETENCLIMIT

Syntax

SETENCLIMIT	**axis [, value]**

Arguments

axis	name of axis device
value	double constant or variable

Description

It changes the incorrect encoder connection limit. This parameter is expressed in the axis UOM. Permitted values must fall within a range equal to 128 – 16384 encoder steps. If the parameter is omitted, the default value equal to 1024 steps is restored.
For example, permitted values for an axis with a 1000 impulse/mm resolution will range from 0.128 to 16.384 mm.

If the **value** parameter is set to zero, the control of the incorrect encoder connection limit is disabled.

Example

```
; set a incorrect encoder connection limit equal to 3.5
SETENCLIMIT X, 3.5
```

SETINDEXEN

Syntax

SETINDEXEN	**axis, state**

Arguments

axis	name of axis device
state	default constant. Permitted values: **ON** zero pulse state enabled **OFF** zero phases pulse disabled

Description

It enables or disables coordinate zeroing on the indicated **axis** at the zero pulse.
To execute this instruction, the axis must be a metering-type axis.

SETINTEGTIME

Syntax

SETINTEGTIME	**axis [, value]**

Arguments

axis	name of axis device
value	integer constant or variable

Description

It sets the number of link error samples used to calculate the integral component. Values are valid from 1 to 200. This parameter may be changed suddenly, but this may generate steps on the axis speed reference. It is advisable to change this parameter when the axes are stationary and disabled, or preferably free.

SETIRMPP

Syntax

SETIRMPP	**axis, speed**

Arguments

axis	name of axis device

speed	float constant or float variable. Ramp start speed

Description

It assigns the *ramp start* **speed** value to the **axis**. It is the minimum speed of a step-by-step motor.
This instruction is used for axes moved by step-by-step motors.

SETLIMNEG

Syntax

SETLIMNEG	**axis [, position]**

Arguments

axis	name of axis device
position	constant or variable. Negative limit

Description

It sets the **axis** negative limit **position**.
If **position** is omitted, the configuration negative limit is set.
These instructions are usually used in homing routines to look for home switches, allowing the axes to exceed set configuration values.
See also instructions RESLIMNEG, SETLIMPOS, RESLIMPOS.

Example

Axis Homing routine

SETLIMPOS

Syntax

SETLIMPOS	**axis [, position]**

Arguments

axis	name of axis device
position	constant or variable. Positive limit

Description

It sets the positive limit **position** for the **axis**.
If **position** is omitted, the configuration positive limit is set.
These instructions are usually used in homing routines to look for home switches, allowing the axes to exceed set configuration values.
See also instructions RESLIMNEG, RESLIMPOS, SETLIMNEG.

Example

Axis Homing routine

SETMAXER

Syntax

SETMAXER	**axis, value [, direction]**

Arguments

axis	name of axis device
value	constant or variable. Maximum loop error
direction	predefined constant. Axis direction Possible values are: **POSITIVE** **NEGATIVE**

Description

It assigns to the **axis** the maximum chase **value** admitted by control, in the indicated direction, before generating a "servoerror".
If **direction** is omitted, the maximum tracking value is set for both directions.

SETMAXERNEG

Syntax

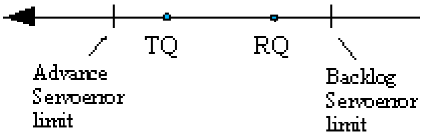
SETMAXERNEG **axis, backlog , advance**

Arguments

axis	name of axis device
backlog	constant or variable. Maximum backlog error
advance	constant or variable. Maximum advance error

Description

Sets the **axis** maximum values for backlog and advance loop errors allowed by control, in negative direction, before generating "servo error". Loop error is computed as the difference between theoretical coordinate (where the axis should be positioned) and real coordinate. When the axis moves in negative direction, a negative value of loop error indicates that the axis has a backlog, while a positive value of loop error indicates that the axis is in advance. If this instruction is not used, the maximum loop error values set in axis configuration will be assumed as default by the numerical control; in this case, the maximum advance error will be equal to 1/4 of the maximum backlog error.



Example

```
;Maximum axis delay is 10mm, maximum advance 5mm
SETMAXERNEG     Axes.X, 10, 5
```

SETMAXERPOS

Syntax

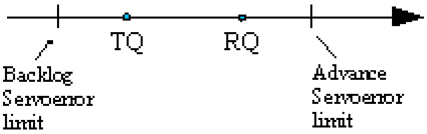
SETMAXERPOS **axis, backlog , advance**

Arguments

axis	name of axis device
backlog	constant or variable. Maximum backlog error
advance	constant or variable. Maximum advance error

Description

Sets the **axis** maximum values for backlog and advance loop errors allowed by control, in positive direction, before generating "servo error". Loop error is computed as the difference between theoretical coordinate (where the axis should be positioned) and real coordinate. When the axis moves in positive direction, a positive value of loop error indicates that the axis has a backlog, while a negative value of loop error indicates that the axis is in advance. If this instruction is not used, the maximum loop error values set in axis configuration will be assumed as default by the numerical control; in this case, the maximum advance error will be equal to 1/4 of the maximum backlog error.



Example

```
;Maximum axis delay is 10mm, maximum advance 5mm
SETMAXERPOS     Axes.X, 10, 5
```

SETPHASESINV

Syntax

SETPHASESINV **axis, state**

Arguments

axis	name of axis device
state	default constant. Permitted values: **ON** phases inversion stage enabled **OFF** phases inversion state disabled

Description

It enables or disables phases inversion on the indicated **axis,** allowing any encoder phase wiring inversion to be offset using software. If used with the reference inversion, the axis direction can be inverted (if wiring is correct).
To execute this instruction, the axis must be in a FREE state.

SETMAXERTYPE

Syntax

SETMAXERTYPE **axis, type**

Arguments

axis	name of axis device
type	integer constant. Permitted values: 0 = sets servoerror to threshold value (default value) 1 = sets dynamic servoerror

Description

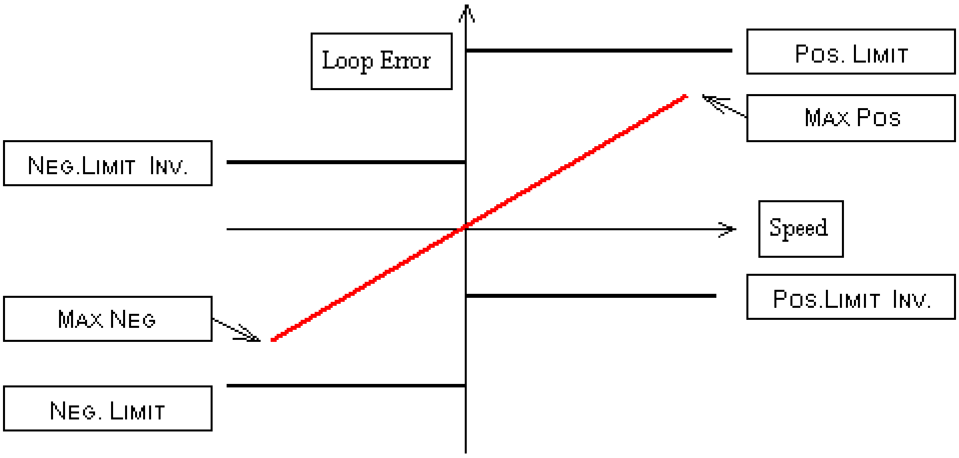
This instruction allows the **type** of servoerror test to be set. Conventional servoerror management sets a pair of limits (positive and negative), which are constant as axis speed changes. This type of management sizes the limits depending on the axis's maximum speed, i.e. it sets a limit so that the error in normal operating conditions is not set off. However at low speeds, the link error generally has far lower values than the set limit, and this delays error condition identification.

Window management of the servoerror is based on calculating the theoretical link error. The positive and negative servoerror limits are calculated as a function of this, adding and subtracting a threshold value from them. If the actual link error exceeds this threshold, a servoerror is generated.

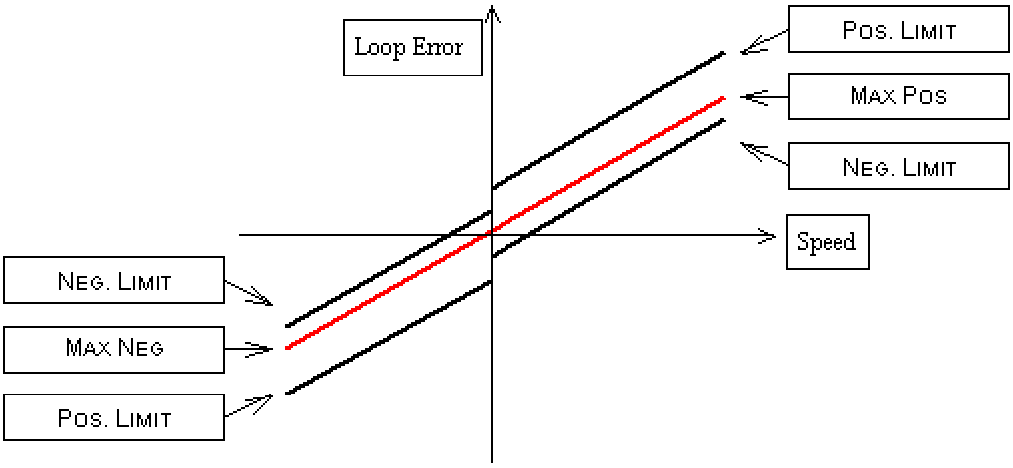
Nota

If you set the test on dynamic servoerror it is generally necessary to amend the limit values of positive servoerror and negative servoerror limit set in axis configuration for the servoerror threshold. This is because the above values are used as initial values for the calculation of the loop-error.

"Classic" ServoError limit":



"Window" ServoError limit":



SETREFINV

Syntax

SETREFINV **axis, state**

Arguments

axis name of axis device

state default constant. Permitted values:
ON reference inversion state enabled
OFF reference inversion state disabled

Description

It enables or disables reference inversion on the indicated **axis.** If used with phases inversion, the axis direction can be inverted (if wiring is correct).
To execute this instruction, the axis must be in a FREE state.
See also SETPHASESINV.

SETRESOLUTION

Syntax

SETRESOLUTION **axis [, value]**

Arguments

axis device name of axis type

value	constant or double variable

Description

changes the resolution of the specified axis. If **value** is left out, the resolution value, that was set in the configuration, is used. Resolution value can only be edited if the axis is stationary (axis state=coordinate), otherwise the system error no. 4101 "Inconsistent axis management" is generated.

3.2.5 Counter

DECOUNTER

Syntax

DECOUNTER **countername [, value]**

Arguments

countername	name of counter device
value	constant or variable or counter device

Description

It decreases the counter **countername** by the specified **value**. If no **value** is set, it assumes value 1. See also instructions SETCOUNTER and INCOUNTER.

INCOUNTER

Syntax

INCOUNTER **countername [, value]**

Arguments

countername	name of counter device
value	constant or variable or counter device

Description

It increases the counter **counter name** by the specified **value**. If no **value** is set, it assumes value 1. See also instructions SETCOUNTER and DECOUNTER.

SETCOUNTER

Syntax

SETCOUNTER **countername, value**

Arguments

countername	name of counter device
value	constant or variable or counter device

Description

It sets the counter **countername** to the specified **value**.
See also INCOUNTER and DECOUNTER.

3.2.6 Timer

HOLDTIMER

Syntax

HOLDTIMER **timername**

Arguments

timername	name of timer device

Description

It blocks the updating of the timer **timername**.
See also STARTTIMER and SETTIMER.

SETTIMER

Syntax

SETTIMER **timername, time**

Arguments

timername	name of timer device
time	constant or variable or timer device

Description

It sets the **timername** to the specified **time** (in seconds).
Only positive values (higher than 0) are admitted. Maximum precision of timers is 4 ms.
See also STARTTIMER and HOLDTIMER.

Example

```
;The Function sets a timer
SETTIMER        Timeout,20      ; Set timer TimeOut to value: 20 seconds
STARTTIMER      Timeout,DOWN    ; Timer starts in decrease mode. When it
                                ; reaches 0 it stops
```

STARTTIMER

Syntax

STARTTIMER **timername [, direction]**

Arguments

timername	name of timer device
direction	predefined constant. Possible values are: **UP** crescent **DOWN** decrescent

Description

It starts the **timername** timer on the mode specified by **direction,** if specified.
If **direction** is omitted, it is automatically set on **DOWN** mode.
When a timer (started in decrescent mode) reaches zero it automatically stops.
See also HOLDTIMER and SETTIMER.

3.2.7 Variables, Vectors and Matrixes

CLEAR

Syntax

CLEAR **varname or vector or matrix[rowmatrix]**

Arguments

varname	name of variable
vector	name of vector
matrix	name of matrix
matrixrow	constant or variable or counter. Matrix row

Description

It clears to 0 the part of memory reserved for variables (**varname**), vectors (**vector**), matrixes (**matrix**) or the elements of a matrix row.

FIND

Syntax

FIND **matrix, column, min_limit, max_limit, value, variable**
FIND **vector, min_limit, max_limit, value, variable**

Arguments

matrix	name of the matrix. The matrix in which to search.
vector	name of the vector. The vector in which to search.

column constant or integer variable or countername. Number of the matrix column in which to search
min_limit constant or variable. Minimum index of the vector or matrix from which search starts
max_limit constant or variable. Maximum index of the vector or matrix where the search ends
value constant or variable. Value to be found
variable variable. Result of the search

Description

It carries out a sequential search of a value inside a **vector** or the **column** of a **matrix** and puts the index of the element in the **variable** variable.
If the value is not found, the **variable** variable will assume value -1.

FINDB

Syntax

FINDB **matrix, column, min_limit, max_limit, value, variable**
FINDB **vector, min_limit, max_limit, value, variable**

Arguments

matrix name of the matrix. The matrix in which to search.
vector name of the vector. The vector in which to search.
column constant or integer variable or countername. Number of the matrix column in which to search
min_limit constant or variable. Minimum index of the vector or matrix from which search starts
max_limit constant or variable. Maximum index of the vector or matrix where the search ends
value constant or variable. Value to be found
variable variable. Result of the search

Description

It performs a rapid search for a value inside a **vector** or the **column** of the **matrix** and puts the index of the element in the **variable** variable. For the search to be successful, the **vector** or the **column** of the **matrix** must have been previously sorted with the SORT instruction according to an increasing order.
If the value is not found, **variable** will assume value -1.

LASTELEM

Syntax

LASTELEM **vector, vectelements**
LASTELEM **matrix, matrows**

Arguments

matrix name of matrix
vector name of vector
vectelements variable. Number of elements of the vector
matrows variable. Number of rows of the matrix

Description

It writes the number of elements of the **vector** in the **vectelements** variable, or the number of rows of the **matrix** in the **matrows** variable.

LOCAL

Syntax

LOCAL **varname AS type**
LOCAL **vector[n° elements] AS type**
LOCAL **matrix[n° rows] AS type, type, type, etc.**
LOCAL **matrix[n° rows] AS type:colname1, type: colname2, type:colname3, etc.**

Arguments

varname name of variable
[n° elements] variable or constant (obligatory argument). Number of elements of the vector
[n° rows] constant or variable (obligatory argument). Number of rows of the matrix
type char, integer (32 bit), float (32 bit), double (64 bit), string, timer
colname1...colnameN name of column. Label.

Description

Declaration of a local variable. Only the PARAM instruction, which defines the parameters of the function, can appear before this instruction.
For further information about local variables see Local variables.

MOVEMAT

Syntax

MOVEMAT matsourcename, mataddrname

MOVEMAT matsourcename[row source], mataddrname[row addr]

MOVEMAT matsourcename[row source], mataddrname[row addr],num row

Arguments

matsourcename name of source matrix
row source start rows number for the copy of the source matrix (obligatory argument)
mataddrname name of addressee matrix
rowaddr start rows number for the copy into the destination matrix (obligatory argument)
numrow rows number to copy

Description

It copies the content of the entire matrix **matsourcename** in the matrix **mataddrname** or one or more rows **num row** of the matrix row **matsourcename[rowsource]** in the matrix row **mataddrname[rowaddr]**. If the parameter **numrow** is not specified one only row is copied. The two matrixes must have the same type of structure (same number of columns and same type of data in each column) and when entire matrix is copied the same number of rows. It is possible to move rows of data within the same matrix.

Example

```
Movemat Mx1, Mx2                 ; copies Mx1 matrix in Mx2

Movemat Mx1[10], Mx2[3]          ; copies row 10 of matrix Mx1 in row 3
                                 ; of Mx2

Movemat Mx1[1], Mx1[7]           ; copies row 1 of matrix Mx1 in row 7
                                 ; of Mx1

Movemat Mx1[2], Mx2[8],6         ; copies 6 rows starting from row 2
                                 : of matrix Mx1
                                 ; into matrix Mx2 starting from row 8
Movemat Mx1[2], Mx1[10],4        ; copies 4 rows starting from
                                 ; row 2 of matrix Mx1 into the same
                                 ; matrix Mx1 starting from row 10
```

PARAM

Syntax

[PARAM] varname AS type
[PARAM] vector[n° elements] AS type
[PARAM] matrix[n° rows] AS type, type, type, etc.
[PARAM] matrix[n° rows] AS type: alias, type:alias, type:alias, etc.

Arguments

varname	name of variable
[n° elements]	constant (obligatory argument)
[n° rows]	constant (obligatory argument)
type	char, integer (32 bit), float (32 bit), double (64 bit), string

Description

The parameters behave like the local variables (see LOCAL), but are activated by whoever calls the function. The syntax for parameter declarations is the same used for local variables.
Parameters may be by value or by reference depending on their kind. See "Functions".
They must be declared before any other instruction.
For further information see Local variables.

SETVAL

Syntax

SETVAL	**value, varname**

Arguments

value	constant or variable or devicename
varname	variable or devicename

Description

It assigns the specified **value** to the **varname** variable or to the n-th vector or matrix element.

SORT

Syntax

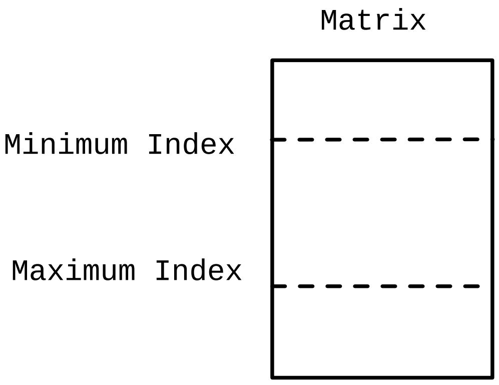
SORT	**matrix, column [, order], min_limit, max_ limit**
SORT	**vector [,order], min_limit, max_limit**

Arguments

matrix	name of the matrix.
vector	name of the vector.
column	constant or integer variable or countername. Matrix column number
order	predefined constant. It indicates order mode Possible values are: **UP** increasing order **DOWN** decreasing order
min_limit	constant or variable. Minimum index of the vector or matrix from which sorting starts
max_limit	constant or variable. Maximum index of the vector or matrix where sorting ends

Description

It sorts the values inside a **vector** or a **matrix**, according to the order specified in the **order** constant.
In the case of a matrix, the order of the rows is dictated by the increasing (UP) or decreasing (DOWN) disposition of the values in the selected **column**.
If the **order** argument is omitted, the UP mode is automatically selected.



3.2.8 Strings

ADDSTRING

Syntax

ADDSTRING	**stringname1, stringname2, stringname3**

Arguments

stringname1	string constant or string variable. Source string
stringname2	string constant or string variable. String to be added
stringname3	string variable. Result string

Description

Chain of two strings.
It adds the string identified by stringname2 to the string identified by stringname1 and puts the result in the string identified by stringname3.
The maximum dimension of a string is 255 characters+ the terminator, so that the result of the chaining of the first two strings can not exceed this limit.

Example

Operations on strings

CONTROLCHAR

Syntax

CONTROLCHAR	**value, stringname**

Arguments

value	char or integer constant or char or integer variable. Value to be converted
stringname	string variable. Result string

Description

It converts the value identified by **value** in ASCII characters and puts the result in the **stringname** string (which corresponds to the first byte).
The former content of the string is lost. This instruction is useful if control or unprintable characters(such as the character NULL = 0x00) have to be inserted in a string.
It accepts strings of at least 2 characters: 1 character + the terminator. If the string is of only one array[1] as char character, the "Incorrect macro argument" system error is signalled

Example

Operations on strings

LEFT

Syntax

LEFT	**sourcestringname, numcharacters, leftstringname**

Arguments

sourcestringname	string constant or string variable. Source string
numcharacters	constant or variable. Number of characters to be copied
leftstringname	string variable. Destination string

Description

It copies the first **numcharacters** of the **sourcestringname** in the **leftstringname**.
In practice, it fetches the left side of the source string. See also instructions MID and RIGHT.

Example

Operations on strings

LEN

Syntax

LEN	**stringname, variable**

Arguments

stringname	string variable. String
variable	variable

Description

It calculates the number of characters contained in the **stringname** string (excluding the terminator) and puts the result in **variable**.

Example

Operations on strings

MID

Syntax

MID	**sourcestringname, firstchar [, numcharacters], rightstringname**

Arguments

sourcestringname	string constant or string variable. Source string
numcharacters	constant or variable. Number of characters to be copied
rightstringname	string variable. Destination string
firstchar	constant or variable. Position of start copy character

Description

It extracts a number of characters identified by **numcharacters**, starting from **firstchar,** from the string identified by **sourcestringname**.

The extracted substring is set in the string identified by namerightstring.

If **numcharacters** is omitted, the **sourcestring** is copied from the **firstchar** position, to the end of it. In practice it fetches the middle part of the source string.

See also instructions LEFT and RIGHT.

Example

Operations on strings

RIGHT

Syntax

RIGHT	**sourcestringname, numcharacters, rightstringname**

Arguments

sourcestringname	string constant or string variable. Source string
numcharacters	constant or variable. Number of characters to be copied
rightstringname	string variable. Destination string

Description

It copies the last **numcharacters** of the **sourcestringname** string in the **rightstringname** string. In practice, it fetches the right side of the source string. See also instructions LEFT and MID

Example

Operations on strings

SEARCH

Syntax

SEARCH	**stringname, character, variable**

Arguments

stringname	string variable.

character	char constant or string constant or string variable. Character or string to be found
variable	variable

Description

It looks for the position of the ASCII character identified by **character** (which may also be a string) within the **stringname** string and puts the index of the result in **variable**.
If **character** is not found, **variable** will contain the value -1.

Example

Operations on strings

SETSTRING

Syntax

SETSTRING **"value", stringname**

Arguments

value	string constant or string variable (in inverted commas)
stringname	destination string

Description

It copies a string.
It copies the ASCII characters contained in the string identified by **"value"** in the string identified by **stringname**.
To insert unprintable characters in a string see instruction CONTROLCHAR.

Example

Operations on strings

STR

Syntax

STR **value, stringname**

Arguments

value	constant or variable. Source value to be converted
stringname	string variable. Destination string

Description

It converts the **value** in ASCII characters and puts the result in the **stringname** string. It can be used to change an integer variable in a string. For example the number 10 becomes the string "10".

Example

Operations on strings

VAL

Syntax

VAL **stringname, result**

Arguments

stringname	string variable. String to be converted
result	variable. Transformed string

Description

It transforms the content of the **stringname** string in a decimal number and puts the result in the **variable**.
For example, the "123" string becomes 123..

Example

Operations on strings

3.2.9 Communications

CLEARRECEIVE

Syntax

CLEARRECEIVE

Arguments

No argument

Description

It empties the list of executed but not satisfied RECEIVES.

COMCLEARRXBUFFER

Syntax

COMCLEARRXBUFFER **COMnumber**

Arguments

COMnumber predefined constant. Number of serial port. Possible values are: from **COM1** to **COM8.**

Description

The instruction empties the receive buffer of the serial **COMnumber**. Any data contained is deleted.

COMCLOSE

Syntax

COMCLOSE **COMnumber**

Arguments

COMnumber predefined constant. Number of serial port. Possible values are: from **COM1** to **COM8.**

Description

It closes the **COMnumber** serial line opened by a **COMOPEN**. It is also necessary to close the serial line when a task that has opened a serial port is closed for any reason.

COMGETERROR

Syntax

COMGETERROR **COMnumber, variable**

Arguments

COMnumber predefined constant. Number of serial port. Possible values are: from **COM1** to **COM8.**

variable integer variable. The result of the last operation executed on the serial

Description

The instruction reads the return code of the last serial communication instruction called on the **COMnumber** port. Through this instruction it can learn whether a read or write task was successful and, if not, it can find the returned error code.
The error codes are listed below.

Normal return	0
Transmission buffer full	2
Device already open	3
Port not valid or not configured	6
I/O port enabling failed	7
Connection to interrupt not possible	8
Serial port (com) not yet open	9
The serial device (com) is occupied	12
Connection to RTX not possible	14

COMGETRXCOUNT

Syntax

COMGETRXCOUNT	**COMnumber, numchar**

Arguments

COMnumber	predefined constant. Number of serial port. Possible values are: from **COM1** to **COM8.**
numchar	number of characters in buffer

Description

The instruction returns the number of characters present in the reception buffer. It allows you to know if the serial port has received any characters.

COMOPEN

Syntax

COMOPEN	**COMnumber, baudrate, wordsize,stopbits,parity**

Arguments

COMnumber	predefined constant. Number of serial port. Possible values are: from **COM1** to **COM8.**
baudrate	communication baudrate. Possible values are: 2400, 4800, 9600, 19200, 38400, 57600, 115200
wordsize	size of data words. Possible values are. 5, 6, 7, 8.
stopbits	stop bits. Possible values are: 1, 2
parity	predefined constant. Parity. Possible values are: **NOPARITY**, **ODDPARITY** and **EVENPARITY**

Description

It opens a serial line. This instruction is executed before any other instruction for serial line management. If any other instruction concerning the same serial line is executed before COMOPEN, a system error is generated. The transmitted parameters must be included among the above mentioned values.
The serial line communication channel is bound to the task wich has executed the COMOPEN instruction. If task ends, the communication channel is automatically closed.
See also COMCLOSE, COMREAD, COMWRITE, COMREADSTRING, COMWRITESTRING.

Note

The number of the serial available lines depends on the hardware environment of the numeric control (see documentation). In the RTX environment only COM1 and COM2 are available.

COMREAD

Syntax

COMREAD	**COMnumber, buffer, numchartoread, numcharread [,timeout]**

Arguments

COMnumber	predefined constant. Number of serial port. Possible values are: from **COM1** to **COM8.**
buffer	vector of char. The vector where the data is deposited.
numchartoread	number of characters which should be read on the serial line
numcharread	number of characters really read
timeout	wait timeout (in seconds)

Description

The instruction reads certain characters of the **COMnumber** serial. The read characters are memorised in the variable **buffer**. This variable must be char vector type. The field **ToRead** indicates the number of characters that the instruction must read. If the serial reception buffer contains less characters and the **timeout** parameter is not specified, the instruction will end immediately, specifying the number of characters it has really read in the parameter **Read**. If the parameter **timeout** is specified, the instruction will have to wait a maximum of seconds indicated in the variable, for other characters to arrive. If **timeout** runs out, the instruction will exit, still specifying in **Read** the number of characters really copied in **buffer**.

COMREADSTRING

Syntax

COMREADSTRING	**COMnumber, buffer, numcharread [,terminator [,timeout]]**

Arguments

COMnumber	predefined constant. Number of serial port. Possible values are: from **COM1** to **COM8.**
buffer	vector of char. The vector where the data is deposited.
numcharread	number of characters really read
terminator	transmission termination character
timeout	wait timeout (in seconds)

Description

The instruction reads certain characters of the **COMnumber** serial. Unlike the **COMREAD** it reads the serial until it finds the terminator character. The read characters are memorised in the variable **buffer** . This variable must be a char type vector. The **numcharread** field indicates the number of characters which the instruction has really read in the serial line and copied in the **buffer**. The parameter **terminator** indicates the character that will function as transmission terminator. In practice the instruction will have to read the characters of the serial until it reaches a character like the one specified in this parameter. This parameter is optional. If no other character is set, the terminator character is zero. The zero is not copied in the buffer as it is recognised as a parameter, while any other termination character specified in the instruction will be copied. The **timeout** is another parameter that indicates how many seconds the instruction will have to wait for more characters if it has emptied the reception buffer without finding any termination character. If the **timeout** parameter is not specified, the instruction will terminate as soon as the reception buffer has been emptied.

COMWRITE

Syntax

COMWRITE	**COMnumber, buffer, towrite**

Arguments

COMnumber	predefined constant. Number of serial port. Possible values are: from **COM1** to **COM8.**
buffer	char vector. The vector containing the data to be written.
towrite	number of characters to be written

Description

The instruction writes the characters present in the buffer variable in the **COMnumber** serial line. The **towrite** parameter specifies the number of characters to be written.

COMWRITESTRING

Syntax

COMWRITESTRING	**COMnumber, buffer [,terminator]**

Arguments

COMnumber	predefined constant. Number of serial port. Possible values are: from **COM1** to **COM8.**
buffer	char vector. The vector containing the data to be written.
terminator	transmission termination character

Description

The instruction writes the characters contained in the buffer variable on the **COMnumber** serial line. Unlike the **COMWRITE** it writes on the serial until it finds the character **terminator**. The parameter **terminator** is optional. If it is not specified, the instruction will transmit until it finds a zero character. The zero is not transmitted, while any other specified control character is.

RECEIVE

Syntax

RECEIVE	**[source,] identifier, flags [, container]**

Arguments

source	string constant
identifier	string constant
flags	integer constant
container	name of device or variable (numeric or string)

Description

This instruction is used, together with SEND, to exchange information between the modules of the plant and the supervisor PC. SEND is used to send information, RECEIVE to ask for information. Information can be requested from Albatros or an external program (Server OLE Automation). In the second case the request is still received by Albatros who will then send it to the external program.

The parameter **source** is a string that allows you to specify where the request for information is directed to. There are three classes of sources:

- sources beginning with the "@" character (see list further on). The source is really Albatros, or better, one of its functions.
- sources not beginning with the "@" character. They are considered as Server OLE, as soon as Albatros receives an information request addressed to them, it will try to send them in execution and then to pass on the information request received from the module.
- unspecified source (the parameter is actually optional). In this case the information is read in a table kept by Albatros. If the information is not included in the table the request remains open and will be satisfied as soon as the information is available (provided by another module or an external program).

The parameter **identifier** is the name of the requested information, and can not be omitted. It takes on different meanings according to the source:

- if Albatros is the source, the identifier will be a command related to the accessed function
- if a Server OLE is the source, it will be a property of the OLE object requested.
- if the source is not specified it will be the label that identifies the information in the Albatros table.

The **flags** parameter allows you to specify how the requested information is to be treated by Albatros. The acceptable values and their effects are the following:

value	***command***	***description***
$0008H	CancelAfter	The information is deleted after being read.
$0800H	UpdateFlags	Modifies the state of the information (already read/to be read) without modifying the data
$8000H	Delete	Deletes the information

The parameter **container** is the variable (or device) in which the requested information will be stored. This may be omitted, in which case the request is the notification of an event (it can be used to synchronise the execution of the GPL code on various modules).

List of **sources** managed by Albatros and their commands:

"@List"

Makes possible to control the commands Simulation and Setpoint
Following commands are allowed (Parameter **identifier**):

- Sim,0,container: requires the Simulation button state, that is written on the Simulazione flag switch. The return variable **container** has a 1 value, if any error did not occur, otherwise it has a value 0.
- Setp,0,container: requires Setpoint button state, that is written on CmdSetP flag switch. The return variable **container** has a 1 value, if any error did not occur, otherwise it has a 0 value.
- Esc,0, container: requires Setpoint button state, that is written on Escluso flag switch. The return variable **container** has a 1 value, if any error did not occur, otherwise it has a value 0.

"@Environ"

It allows you to receive information about the state of the system: user's access level, modules connected to the supervisor etc. The requested information is stored in the parameter **container**. The acceptable values for the parameter **identifier** and the relative answers are:

• AccessLevel"	access level to the system 0=user, 1=service, 2=builder, 3=tpa
• "MaskConfModules"	mask of configured modules
• "MaskActiveModules"	mask of connected modules
• "CurrentModule"	module sending the request
• "*mod*:NamePC"	name of PC corresponding to module "*mod*". (*mod* must be between 0 and 15)
• "LocalDateTime"	date and time of PC in YYYY/MM/DD HH:MM:SS format

The masks of the connected and configured modules are bit masks. The lowest weight bit is module 0. The bit of each module is 1 if the module is connected or configured. In case of "NamePC" the module number is not compulsory; if omitted, the number is assumed of the module which instanced the request.

"@Syn"

Communication between GPL and the synoptic view display. It allows you to open and close the synoptic views with GPL control and request information from a synoptic cell.
The following commands are possible (parameter **identifier**):

- "Open:*filename*" opening of the synoptic *filename.syn*
- "Close:*filename*" closure of the synoptic *filename.syn*
- "*cellname*" cell from which the requested information is read

It is possible to get information about the axes move window according to the technical data, that has been defined also for the parameter **source** "@Devices" , as below.
It is possible to get some information about the axes movement, according to the specifications defined also for the parameter **source** "@Devices", as below.

"@FileName"

stores an association between a constant string and a file name, which can be made up with string variables. Since Albatros has received the communication of the association it replaces all the following file names with the name received by means of this instruction. The parameter **identifier** is the name of the file. The name of the file is a variable string. If in the parameter identifier the complete path in which to store the file is not specified, Albatros considers the one defined in tpa.ini into the section [tpa] at the item *dirreport*. The value of the parameter identifier is stored in tpa.ini in the section [GPLFileName] at the item Log, so that it can be used again also in the Albatros executions, that follow. To cancel the association you need to set an empty string as parameter identifier. The association, which is defined in this way, can be used for each module.

"@FileDelete"

Delete a file. The **identifier** parameter is the name of the file which will be deleted (complete path). If in the parameter identifier the complete path in which to store the file is not specified, Albatros considers that defined in tpa.ini in the section [tpa] at the item *dirreport*. The file name can be defined according to the rules, that have been described in the parameter **source** @FileRead. The **container** parameter contains the value:

- 1 if the file has been deleted
- 0 if not

"@FileRead"

It reeds the file content. The parameter **indentifier** is the name of the file that will be read (complete path). If in the parameter identifier the complete path in which to store the file is not specified, Albatros considers that defined in tpa.ini in the section [tpa] at the item *dirreport*. If the identifier starts and finishes with a %-symbol, the inside string is searched in tpa.ini into the section [tpa] and used as a file name. Inside the name can be inserted some symbols that will be substituted during the instruction execution:

- %n module number that execute RECEIVE instruction
- %h current time (format 00-23)
- %d current day (format 01-31)
- %m current month (format 01-12)
- %y current year (four numbers format)

If the parameter **container** is defined as a char variable, it will contain a byte read by the file, if it is define as a string, it will contain an entire string of the file test, if defined as a file integer, it will contain the missing number of bytes to reach the end of the file (0= file end).
To place the pointer on the file at the beginning of the file itself, the parameter **container** should be omitted.

"@FileExist"

It checks the existence of a file. The parameter **identifier** is the name of the file that will be read (complete path). If in the parameter identifier the complete path in which to store the file is not specified, Albatros considers that defined in tpa.ini in the section [tpa] at the item *dirreport*. The name of the file can be defined according to the rules that have been described in the parameter **source** @FileRead. The parameter **container** contains the value:

- different from 0, if the file exists
- 0, if the file does not exist

"@Devices"

Request to open or close the Diagnostic window of the module sending the information. The identifier parameter can assume the following values:

- "Open" open Diagnostic
- "Close" close Diagnostic

The parameter **identifier**, when we need to interact with the move axis window, can assume the values, as follows:

- "MoveAX#nome_asse#HasFocus" the parameter **container** contains 1, if the specified move axis window is active, otherwise it contains 0.
- "MoveAX#nome_asse#Jog" the parameter **container** contains 1, if the move for displacements managed in runtime by the operator is set, otherwise it contains 0.
- "MoveAX#nome_asse#Step" the parameter **container** contains 1, if the move with predefined steps is set, otherweise it contains 0.
- "MoveAX#nome_asse#Absolute" the parameter **container** contains 1, if the move with defined position is set, otherweise it contains 0, where the axis name represents the name of the axis displayed in the window. E.g., if we need to verify, if the move axis window is active, the parameter **identifier** will be "@MoveAX#X#HasFocus". The name of the axis can be expressed in one of the following forms:

1. Name_Group. Name_Subgroup. Name_Axis or Name_Group. Name_Axis: the complete path of the axis is shown.
2. Name_Axis: to identify the correct axis checks are made according to the following order:
    - if the task from which it arrives the command is a function of subgroup, the axis is searched in that sub-group.
    - if the task from which it arrives the command is a function of the main subgroup, the axis is searched in all the group. If there is more than one axis with that name, the research fails.
    - if the previous checks failed, the axis is searched in all the groups in the module. If there is more than one axis with the name Name_Axis, the research has not positive outcome.

"@Vars"

It requests the updating of a GPL global variable. It allows you to perform data refreshment of technological Parametric and tools. The parametric data is normally sent to the GPL during machine booting. The parameter **identifier** will indicate the name of the global variable (machine or group) whose update is requested. The parameter **container** will contain the value:

- 1 if the variable has been correctly updated
- 0 if not

"@Application"

Interaction with Albatros. It allows you to display the "message box" on the screen and close down Albatros. Possible values for the **identifier** parameter are:

- "Quit" to close Albatros
- "IsLocked" verifies, if the exit from the Albatros is locked. The parameter **container** contains 1, if the interface is locked, 0, if it is possible to exit Albatros.
- "MsgBox" reads the answer of a message box previously opened with a SEND

The parameter **container** makes it possible to know, in the case of a message box, which button has been pressed by the operator:

- 1 "OK" button
- 2 "Cancel" button
- 3 "Abort" button
- 4 "Retry" button
- 5 "Ignore" button
- 6 "Yes" button
- 7 "No" button

In the case of the "Quit" control, the parameter **container** will contain the value:

- 1 IF Albatros has been closed down correctly
- 0 if not

"@Param"

It allows you to know the progressive numer of Partec.par and Partool.par parametric files storing. Requested information is stored into **container** parameter. Admited values for the parameter **identifier** are:

- "partec" it requests the progressive of partec.par storing
- "partool" it requests the progressive of partool.par storing

"@Ini"

reads a key=value combination from the tpa.ini file. The parameter **identifier** is the name of the key to read in tpa.ini at section [Tpa]. To read from a specific section, the name of the section in square brackets ("[Section]Key") must be added to the name of the key.

"@ShellExecute"

asks the operating system to open a file using the program associated to the file extension. An executable program can be also launched. The parameter **identifier** is the name of the file to open or the name of the program to launch. The name of the file can be declared with a complete path; if not, it is charged in the current folder of Albatros. The name of the file is searched also among those, that are defined through "@FileName". The parameter **container** contains the value 0, if no errors occurred while opening the file; otherwise, it contains the code of the error.

"@StartProg"

execute the program defined in the parameter **identifier**. In is not possible to pass the arguments to the program to launch. The name of the program must contain the whole path; if not, it is charged in the current folder of Albatros. The name ofthe program is also searched also among those that are defined through "@FileName". The parameter **container** contains the value 0, if the program was successfully launched; otherwise, it contains the code of the error. If the program had already been launched, the code or the error is 1056.

"@ProgRunning"

verifies if the program, launched with "@StartProg" is still being executed. The name of the program is defined in the parameter **identifier**. The name of the program must contain the whole path; if not, it is charged in the current folder of Albatros. The name ofthe program is also searched also among those that are defined through "@FileName". The parameter **container** contains value 1, if the program is still being executed, if not it contains value 0.

"@TermProg"

ends the program defined in the parameter **identifier** and launched through "@StartProg" . The name of the program must contain the whole path; if not, it is charged in the current folder of Albatros. The name of the program is searched also among those, that are defined through "@FileName". The parameter **container** contains the value 0, if the program was successfully launched; otherwise, it contains the code of the error. If the program had already been launched, the code or the error is 1056.

"@DialogFile"

opens the dialog box of File Open or File Save to allow you to choose a file name. To open the window of File Open set the parameter **identifier** = "Open", to open the window of File Save to set the parameter **identifier** = "Save". The name of the selected fileis stored in the parameter **container.**

Example

```
 ;in GPL
   RECEIVE "@Param", "partec", 0, prog
   RECEIVE "@Param", "partool, 0, prog

;in GPL
; reads the Radix key value in the [Albatros] section from the
;tpa.ini file
   RECEIVE "@INI", "[Albatros]Radix", 0, value

; opens the window of File Open and stores the name of file in the FileName
                                variable
RECEIVE "@DialogFile", "Open", 0, FileName
```

SEND

Syntax

SEND **[addressee,] identifier, flags [, information]**

Arguments

addressee	string constant
identifier	string constant
flags	integer constant
information	name of device or constant or variable (numeric or string)

Description

This instruction is used, together with RECEIVE, to exchange information between the modules of the plant and the supervisor PC. SEND is used to send information, RECEIVE to ask for information. Information can be requested from Albatros or an external program (Server OLE Automation). In the second case the request is still received by Albatros who will then send it to the external program.

The parameter **addressee** is a string which allows you to specify who the information is sent to. There are three classes of addressees:

- addressees beginning with the "@" character (see list further on). The addressee is really Albatros, or better, one of its functions.
- addressees which do not begin with the "@" character. They are considered as Server OLE, and as soon as Albatros receives an information request addressed to them, it will try to send them in execution and then to pass on the information request received from the module.
- unspecified addressee (the parameter is actually optional). In this case the information is kept in a table by Albatros where it is available for anyone requesting it (another module or external program).

The parameter **identifier** is the name of the information, and can not be omitted. It takes on different meanings according to the addressee:

- if Albatros is the addressee, the identifier will be a command related to the accessed function
- if a Server OLE is the addressee, it will be a property of the OLE object requested.
- If the addressee is not specified it will be the label identifying the information contained in the Albatros table

The parameter **flags** allows you to specify how the requested information is to be treated by Albatros. The acceptable values and their effects are the following:

value	***command***	***description***
$0001H	Broadcast	Normal request broadcast
$0008H	CancelAfter	The information is deleted after being read.
$0020H	ReadOnly	The information can only be deleted by the sender
$1000H	UpdateFlags	Modifies the state of the information (read / to read) without modifying the data
$8000H	Delete	Deletes the information

The **information** parameter is the information sent. This can be omitted, in which case the empty information indicates the notification of an event (it can be used to synchronise the execution of the GPL code on a series of modules). All devices (except for the axes), simple GPL variables and strings are recognised as information parameters.

List of **addressees** managed by Albatros and their commands:

"@List"

makes possible to control the commands Simulation and Setpoint
Following commands are allowed (parameter **identifier**):

- Sim: notifies the change in state of the Simulating switch flag. According to the flag state, its identification button is visualized pressed or released in the toolbar (1=checked, 0=unchecked).
- Setp: notifies the change in state of the CmdSetp switch flag. According to the flag state, its identification button is visualized pressed or released in the toolbar (1=checked, 0=unchecked).
- Esc: notifies the change in state of the Excluded switch flag. According to the flag state, its identification button (same as the flag switch CmdSetp button) is visualized pressed or released in the toolbar (1=checked, 0=unchecked)
- End: ends the list execution. This command lowers the Start and Stop buttons and disallows

the Start and Stop options of the menu
- Hold: lowers the Stop button and enables the Stop option of the menu. It raises the Start button and disallows the Start option of the menu

"@Syn"

Communication between GPL and the synoptic view display. It allows you to open and close the synoptic views through GPL control and to send information to a synoptic cell.
The following commands are possible (parameter **identifier**):

"Open:*filename*"	opening of the synoptic *filename.syn*
"Close:*filename*"	closure of the synoptic *filename.syn*
"Open"	opening of a synoptic. The file name is read from variable **information**
"Close"	closure of a synoptic. The file name is read from variable **information**
"*cellname*"	cell in which the sent information is displayed

It is possible to interact with the axis move window according to the technical data, that has been defined also for the parameter **addressee** "@Devices" , as below.

"@File"

Writing on a file. It allows you to create personalised log files to memorise the operations performed by a machine. The files are text files (ASCII). The **identifier** parameter is the name of the file which will be written on.
If in the parameter identifier the complete path, in which to store the file, is not specified, Albatros considers that defined in tpa.ini in the section [tpa] at the item *dirreport*.
If the identifier starts and finishes with the symbol % inside the string is cherched in tpa.ini in section [tpa] and used as file name. Inside the name can be inserted symbols that will be substituted during the instruction execution:

- %n module number that execute SEND instruction
- %h current time (00-23 format)
- %d current day (01-31 format)
- %m current month (01-12 format)
- %y current year (four numbers format)

See the example.
Writing operations are carried out in append mode (the data is added at the end of the file). Numeric data (automatically converted to ASCII) and strings can be sent in a file. It is possible to write date/time format strings using format characters %d for the date and %t for the time. For the time we use the format "HH:mm:ss" (that is: hours, minutes and seconds separated by ":") and for the date we use a format, that depends on each national settings. It is possible to use another format, if you set in tpa.ini in the section [Albatros] the option "LogNoLocale=1" (by default it is LogNoLocale=0, that is use of the current format). It is also possible to set the format to be used for the date and the time apart from the format set in Windows, defining always in tpa.ini in the section [Albatros] the options "LogDateFormat=" e "LogTimeFormat=" and assigning a string of characters according the table below. If these options are not available or are empty, we use the formats set by Windows.

Time format

h	Time in 12-hours format without leading zeros
hh	Time in 12-hours format with leading zeros
H	Time in 24-hours format without leading zeros
HH	Time in 24-hours format with leading zeros
m	minutes without leading zeros
mm	minutes with leading zeros
s	seconds without leading zeros
ss	seconds with leading zeros
t	one only character to show the time marker, e.g. A or P
tt	several characters to show the time marker, e.g. AM or PM

Notes "t" and "tt" format use the time marker shown in the control panel of the current user. It is not necessarily "AM" and "PM".
Example: if it is 11:29 in the afternoon and the string is made up in this way "hh':'mm':'ss tt", "11:29:40 PM" appears.

Day format

d	day of the month without leading zeros, represented by the digits
dd	day of the month with leading zeros, represented in digits
ddd	day of the week, represented in characters and shortened to three letters
dddd	day of the week, represented in characters with its full name
M	month without leading zeros, represented in digits

MM	month with leading zeros, represented in digits
MMM	month, represented in characters and shortened to three letters
MMMM	month, represented in characters with its full name
y	year with two digits without leading zeros for years less than 10
yy	year with two digits with leading zeros for years less than 10
yyyy	year represented by four or five digits according to the calendar in use
yyyyy	year represented by four or five digits according to the calendar in use

Example: if it is Wednesday, 31 August, 1994 and its string is made up in this way "ddd',' MMM dd yy", "Wed, August 31 94" appears.
If the information is omitted a "return to beginning" is added to the file.

"@FileName"

stores an association between a constant string and a file name, which can be made up with string variables. Since Albatros has received the communication of the association it replaces all the following file names with the name received by means of this instruction. The parameter **identifier** is the name of the file, which will be written. The name of the file is a variable string. If in the parameter identifier the complete path in which to store the file is not specified, Albatros considers the one defined in tpa.ini into the section [tpa] at the item *dirreport*. The value of the parameter identifier is stored in tpa.ini in the section [GPLFileName] at the item Log, so that it can be used again also in the Albatros executions, that follow. To cancel the association you need to set an empty string as parameter identifier. The association, which is defined in this way, can be used for each module.

"@FileDelete"

deletes a file. The parameter **identifier** is the name of the file which will be deleted (complete path). If in the parameter identifier the complete path, in which to store the file, is not specified, Albatros considers that defined in tpa.ini in the section [tpa] at the item *dirreport* cannot be used. File name can be defined according to the rules described for the parameter **addressee** @File

"@FileRead"

places the pointer at the beginning of the file. The parameter **identifier** is the file name (complete path). If in the parameter identifier the complete path, in which to store the file, is not specified, Albatros considers that defined in tpa.ini in the section [tpa] at the item *dirreport*. File name can be defined according to the rules described for the parameter **addressee** @File.

"@Axis"

interacts with the axis manual movement window according to the technical data, that have been defined also for the parameter addressee "@Devices", as below. If a window that controls the movements of the indicated axis is already open, this command acts on this window, whether it is open in a synoptic data table or it is open in diagnostics. If the window is shut, the command tries to open it in Diagnostics or in one of the synoptic data tables already open and that contains that axis.

"@Devices"

requires to open or close the Diagnostic window of the module sending the information. Commands execution within the axis move window in diagnostic. The **identifier** parameter can assume the following values:

- "Open" open Diagnostic
- "Close" close Diagnostic

The parameter **identifier**, when we need to interact with the move axis window, can assume the values, as follows:

- "MoveAX#nome_asse#Open" opening of the axis move window
- "MoveAX#nome_asse#Close" closing of the axis move window
- "MoveAX#nome_asse#Plus" pushing the button of axis move (positive direction)
- "MoveAX#nome_asse#Minus" pushing the button of axis move (negative direction)
- "MoveAX#nome_asse#Stop" pushing the button of move stop
- "MoveAX#nome_asse#Jog" setting the mode of move for displacements managed in runtime by the operator
- "MoveAX#nome_asse#Step" setting the mode of move for displacements with predefined steps
- "MoveAX#nome_asse#Absolute" setting the mode of move with axis defined position

where the axis name represents the axis name displayed in the window. E.g, if you need to open the X-axis move window, the parameter **identifier** is "@MoveAX#X#Open". The axis can be named as follows:

1. Name_Group.Name_Subgroup.Name_Axis or Name_Group .Name_Axis: the complete axis path is given.

2. Name_Axis: to identify the right axis, tasks are verified according the following order:
   - If the task from which the command arrives is a function of subgroup, the axis is searched in that subgroup.
   - If the task from which the command arrives is a function of the main group, the axis is searched in all the group. If there is more than one axis with that name, the research fails.
   - If the previous checks failed, the axis is searched in all the groups of the module. If there is more than one axis with Name_Axis, research has not positive outcome.

It is possible to prevent the user to act on the keys of axis move of all the axis movement windows of the module in diagnostic. For this purpose the parameter **identifier** should be set as follows:

- "MoveAX##UIENABLE" if the parameter **information** is set on 0, the axes move from Albatros is disabled; if it is set on 1e, the axes move is enabled from Albatros.

We suggest to disable axes move from Albatros, when the axes are moved from the machine's control panel.

"@Vars"

requires to save the content of a GPL global variable in the store of the technological parameters or tools. The parameter **identifier** is the name of the global variable (of machine whether group or library) for which the update is required.

"@Application"

Interaction with Albatros. It allows you to display "message boxes" on the screen and close down Albatros. Possible values for the **identifier** parameter are:

• "Quit"	to close Albatros
• "Lock"	prevents from closing Albatros from ***File->Exit*** or from keyboard shortcuts [ALT +F4] or from closing button.
• "Unlock"	restores the possibility of closing Albatros
• "MsgBox:*flags*"	to open a message box

The behaviour of the message boxes is controlled by the "*flags*" of the **identifier** string. This can be a combination of the following strings:

• "O"	"OK" button
• "OC"	"OK" and "Cancel" buttons
• "YN"	"Yes" and "No" buttons
• "YNC"	"Yes", "No" and "Cancel" buttons
• "RC"	"Retry" and "Cancel" buttons
• "ARI"	"Abort", "Retry" and "Ignore" buttons
• "S"	Stop icon
• "?"	Question mark icon
• "!"	Exclamation mark icon
• "*"	information icon
• "1"	the first button is for default
• "2"	the second button is for default
• "3"	the third button is for default

For example "MsgBox:?YN2" identifies a message box with a question mark icon and two "Yes" and "No" buttons where the latter one is the default button.
The **information** parameter can be a string, containing the text to be displayed, or an integer number which is recognized as the code of a module message handled by Winmess.exe or a group message label defined by the DEFMSG instruction.

"@Help"

opens a help file. It allows you to command the display of a help file by specifying the argument to be displayed. Possible values for the **identifier** parameter are:

• "Open:*filename*"	to open a help file
• "Close:*filename*"	to close a help file

The "*filename*" part of the string, specifies the name of the help file to be opened.
The parameter **information** can be a string or a number and assumes accordingly the meaning of key or context number (to identify the page or help argument to be displayed).

"@Report"

adds messages to the Albatros report file (MONTH (n month).TER). The parameter **Identifier** is:

- "Add"

The parameter **Information** can be:

- a string variable or a string constant: the text, contained in the string, is saved in the report file
- an integer variable or an integer numeric value: the text, defined by the DEFMSG instruction, is saved
- defined by the DEFMSG instruction.

"@Ini"

writes a key=value combination from the tpa.ini file. The parameter **identifier** is the name of the key to add in tpa.ini at section [Tpa]. To write in a specific section, the name of the section in square brackets ("[Section]Key") must be added to the name of the key
The parameter **information** can be a string or numeric variable, a string or a numeric constant.

"@ShellExecute"

asks the operating system to open a file using the program associated to the file extension. It is also possible to launch an executable program. The parameter **identifier** is the name of the file to open or the name of the program to launch. The name of the file can be declared with a complete path; if not, it is charged in the current folder of Albatros. The name of the file is also searched among those that are defined through "@FileName".

"@StartProg"

executes the program defined in the parameter **identifier**. It is not possible to pass any arguments to the program to launch. The name of the program must contain the whole path; if not, it is searched in the current folder of Albatros. The name of the program is searched also among those that are defined with "@FileName".

"@TermProg"

ends the program defined in the parameter **identifier** and launched through "@StartProg" . The name of the program must contain the whole path; if not, it is charged in the current folder of Albatros. The name of the program is searched also among those that are defined through "@FileName".

"@DialogFile"

allows you to set some parameters related to the dialog box of File Open or File Save.
The values allowed for the parameter **identifier** are:

"Extension"	if the user does not enter an extension, the extension defined in the**information** parameter is used (variable or string constant)
"Filter"	sets the filter on the file types to be used. The **information** parameter can be a string variable or a string constant; in this case the text in the string, an integer variable or an integer numerical value is used as a filter and in this case the text defined in the DEFMSG instruction is used as a filter.
"Flags"	set the initialisation flags. For the list of the values to be set in the **information** field (variable or integer constant), please make reference to the official Microsoft documentation concerning the Flags member of the OPENFILENAME structure.
"InitalDir"	set the initial folder, defined in the **information** field (variable or string constant)
"Title"	sets the box name. The **information** parameter can be a string variable or a string constant; in this case the text in the string, an integer variable or an integer numerical value is used as a filter and in this case the text defined in the DEFMSG instruction is used as a filter.

```
; Example of send file instruction with name created during execution.
: Suppose that the date of instruction execution be 31-01-2000

; in GPL
    SEND "@File", "%Log%", 0, "Start execution"
    SEND "@File", "%Log%", 0                          ; add a "wordwrap"
; in tpa.ini at section  [TPA]
Log=c:\Albatros\report\%y\Rep%m%d.txt


; The name of final file is:
c:\Albatros\report\2000\Rep0131.txt
```

```
; Example of send Vars instruction
; we define a Var_SendVars variable as double in the file of the global
; variables
; in the technological Parameters Var_SendVars is entered in the field
; Matrix Name
; in GPL
     SETVAL  100.0,Var_SendVars
; sends the 100.0 value to the parameter of the technological Parameters
; associated to the Var_SendVars variable
     SEND "@Vars", "Var_SendVars", 0

; Example of send INI instruction
; in tpa.ini the Radix key is entered in the [Albatros] section to set
; a numerical basis of decimal number view
 SEND "@INI", "[Albatros]Radix", 0;1

; Example of setting up an association between GPL constant string
; and name of a file.

; declaration of a string variable
nomefile as string
; composition of the file name
setstring C:\ALBATROS\MOD.0\CONFIG),filename
; association
 SEND "@File", "LOG",0,filename
; all the writing operations from now are
; performed in the file defined by the filename variable
 SEND "@File", "LOG",0, "Writing in the LOG file"
```

SENDIPC

Syntax

SENDIPC	**IPCname, wait [, varname1 [, varnameN, ...]]**
SENDIPC	**IPCname, wait , matrix[row]**
SENDIPC	**IPCname, wait , vector**
SENDIPC	**IPCname, wait , matrix**

Arguments

IPCname	string constant. Name of the IPC
wait	predefined constant. Wait mode of command read Possible values are: **WAIT** waits for the command to be read **NOWAIT** does not wait for the command to be read
varname1[...varnameN]	constant or variable. Names of variables 1÷N
matrix[row]	constant or integer variable. Matrix row number
vector	name of vector
matrix	name of matrix

Description

It sends an IPC command to the "**IPCname**" shared memory.
When the SENDIPC instruction is executed for the first time the shared memory is allocated; the memory's dimension is calculated on the basis of the size of sent data. The maximum shared memory dimension is 64 Kb. Up to 48 shared memories can be defined with 48 distinct names.
A semaphore is connected to the memory to allow synchronisation of the tasks accessing it. The task writing the data enables the semaphore when it finishes writing, the task reading the data disables it when it finishes reading.
If WAIT was indicated as **wait** parameter, the task sending the data will wait for them to be read (disabled semaphore) before continuing execution.
A SENDIPC without data simply synchronises the tasks. In this case no shared memory is allocated.

IPC intermodule

Two remote modules can exchange data through IPCs. These IPCs are called IPC intermodule. To define an IPC intermodule you need to write the **IPCname** according to the following formalism: Number of source module, "->", number of the recipient module, ":", and hereafter the other character of the IPC name.
For example, "0->1:Base Parameters".

See also WAITIPC and TESTIPC.

WAITIPC

Syntax

WAITIPC	**IPCname [, varname1 [, varnameN, ...]]**
WAITIPC	**IPCname, matrix[row]**
WAITIPC	**IPCname, vector**
WAITIPC	**IPCname, matrix**

Arguments

IPCname	string constant. Name of IPC
varname1[...varnameN]	constant or variable. Names of variables 1÷N
matrix[row]	constant or integer variable. Matrix row number
vector	name of vector
matrix	name of matrix

Description

It receives an IPC command from the "**IPCname**" shared memory.
When the SENDIPC instruction is executed for the first time the shared memory is allocated; the memory's dimension is calculated on the basis of the size of sent data. The maximum shared memory dimension is 64 Kb. Up to 48 shared memories can be defined with 48 distinct names.
A semaphore is connected to the memory to allow you to synchronise the execution of the tasks accessing it. The task reading the data waits for the semaphore to be enabled by the task writing the data, it reads the data and then disables the semaphore.

A WAITIPC without data simply synchronises the tasks. In this case the shared memory is not allocated.
See also SENDIPC and TESTIPC.

WAITRECEIVE

Syntax

WAITRECEIVE	**[source,] identifier, flags [, container]**

Arguments

source	string constant
identifier	string constant
flags	integer constant
container	name of device or variable (numeric or string)

Description

It waits for the requested information (specified by identifier) to arrive, before continuing execution of the GPL program. For use, consult documentation of the RECEIVE instruction.

3.2.10 Mathematics

ABS

Syntax

ABS	**operand, result**

Arguments

operand	constant or variable or name of device
result	variable or name of device

Description

It extracts the absolute value of **operand** and puts in **result**. To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
SETVAL    -10,op       ; sets -10 to the op variable
```

```
ABS       op,var

;The value set in the var variable is 10
```

ADD

Syntax

ADD **operand1, operand2, result**

Arguments

operand1	constant or variable or name of device
operand2	constant or variable or name of device
result	variable or name of device

Description

It sums **operand1** to **operand2** and puts the result in **result**. To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
SETVAL    5,op1       ; sets 5 to the op1 variable
ADD       op1,3,var

;The value set in the var variable is 8
```

AND

Syntax

AND **operand1, operand2, result**

Arguments

operand1	constant or variable or name of device
operand2	constant or variable or name of device
result	variable or name of device

Description

It performs a binary AND operation (*between two bits, the result is 1 only if both equal 1*) between **operand1** and **operand2** and puts the result in **result.** To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
;The value set in the var variable is 1
;(Binary notation: 5 = 0101, 3 = 0011, 1 = 0001)

AND 5,3,var
```

ARCCOS

Syntax

ARCCOS **operand, result**

Arguments

operand	constant or variable or name of device
result	variable or name of device

Description

It carries out an arc cosine operation on **operand** and puts the value, in degrees, in **result**. The value of the result can range between 0°÷180°. To convert data, according to the type of declared data, see chapter Data conversion.

ARCSIN

Syntax

ARCSIN	**operand, result**

Arguments

operand	constant or variable or name of device
result	variable or name of device

Description

It carries out an arc sinus operation on **operand** and puts the value, in degrees, in **result**. The value of the result can range between -90°÷+90°. To convert data, according to the type of declared data, see chapter Data conversion.

ARCTAN

Syntax

ARCTAN	**operand1 [, operand2], result**

Arguments

operand1...[operand2]	constant or variable or name of device
result	variable or name of device

Description

If **operand2** is omitted, it carries out an arc tangent operation of **operand1** and puts the value, in degrees, in result.
If **operand2** is present, the considered angle is the one whose sinus is given by **operand1** and whose cosine is given by **operand2**. To convert data, according to the type of declared data, see chapter Data conversion.

COS

Syntax

COS	**operand, result**

Arguments

operand	constant or variable or name of device
result	variable or name of device

Description

It carries out a cosine operation on **operand** and puts the value in **result**.
The argument **operand** is expressed in degrees with a possible centesimal fractionary part (ex.: 30° 15" = 30,25.). To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
SETVAL    60,op       ; sets 60 to the op variable
COS       op,var

;The value set in the var variable is 0.5
```

DIV

Syntax

DIV	**operand1, operand2, result**

Arguments

operand1	constant or variable or name of device
operand2	constant or variable or name of device
result	variable or name of device

Description

It performs a division between **operand1** and **operand2** and puts the result in **result**. The instruction can generate a system error when **operand2** equals 0. To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
SETVAL    10,op1       ; sets 10 to the op1 variable
SETVAL    5,op2        ; sets 5 to the op2 variable
DIV       op1,op2,var

;The value set in the var variable is 2
```

EXP

Syntax

EXP	**operand, result**

Arguments

operand	constant or variable or name of device
result	variable or name of device

Description

It calculates the exponential of **operand** and puts the value in **result**. To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
SETVAL    2.302585093,op    ;sets 2.302585093
                            ;in the op variable
EXP       op,var

;The value set in the var variable is 10
```

EXPR

Syntax

EXPR	**variable = expression**

Arguments

variable	name of device or variable
expression	group of operators

Description

This instruction allows you to resolve mathematical expressions. Factors may be constants, names of devices or variables. Its syntax provides that between each operator and each operand a spacing should be entered.
If the operands are not of the same type, an automatic conversion is carried out and the type of the result of the operation is the same as the greater one, according the following rule:

- char <integer
- float < double
- char or integer < float or double.

After resolving the **expression,** the result is converted to the **variabile** type.

The following operators are permitted:

()	brackets
-	sign change operator
ABS	absolute operand value
ROUND	unit round up/round down
TRUNC	value truncated to whole number
LOG	natural logarithm
LOGDEC	decimal base logarithm
EXP	exponential
SRQ	square root operation

SIN	sine operation. The operand is expressed in degrees, indicating the value to two decimal points if applicable (e.g..: 30° 15" = 30.25.)
COS	cosine function operation. The operand is expressed in degrees, indicating the value to two decimal points if applicable (e.g..: 30° 15" = 30.25.)
TAN	tangent operation, expressed in degrees.
ARCSIN	arc sine operation. The result is expressed in degrees, with the value in a -90°÷+90° range
ARCCOS	arc cosine operation. The result is expressed in degrees, with the value in a 0°÷180° range
ARCTAN	executes an arc tangent operation. See ARCTAN
^	power operator

*	multiplication
/	division
%	division remainder (module)
+	addition
-	subtraction

This instructions allows for GPL code writing to be simplified, when performing mathematical calculations; the single GPL instructions corresponding to the operators listed in the table are replaced. These instructions stay available for compatibility purposes.

Example

```
 ; calculation of the distance between two points

EXPR dist = SQR ( ( Xb - Xa ) ^ 2 + ( Yb - Ya ) ^ 2 )
```

LOG

Syntax

LOG **operand, result**

Arguments

operand	constant or variable or name of device
result	variable or name of device

Description

It calculates the natural logarithm of **operand** and puts the result in **result**. To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
SETVAL    10,op        ; sets 10 to the op variable
LOG       op,var

;The value set in the var variable is 2.302585093
```

LOGDEC

Syntax

LOGDEC **operand, result**

Arguments

operand	constant or variable or name of device
result	variable or name of device

Description

It calculates the base 10 logarithm of **operand** and puts the value in **result.** To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
SETVAL    10,op        ; sets 10 to the op variable
LOGDEC    op,var

;The value set in the var variable is 1
```

MOD

Syntax

MOD **operand1, operand2, result**

Arguments

operand1	constant or integer variable or name of device
operand2	constant or integer variable or name of device
result	integer variable or name of device

Description

It performs a module operation between **operand1** and **operand2** and puts the result in **result**. The module is the remainder resulting from the division between the first and the second operand. The instruction can generate a system error when **operand2** equals 0. To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
SETVAL    20,op1       ; sets 20 to the op1 variable
SETVAL    3,op2        ; sets 3 to the op2 variable
MOD       op1,op2,var

;The value set in the var variable is 2
```

MUL

Syntax

MUL **operand1, operand2, result**

Arguments

operand1	constant or variable or name of device
operand2	constant or variable or name of device
result	variable or name of device

Description

It performs a multiplication operation between **operand1** and **operand2** and puts the result in **result**. To convert data, according to the type of declared data, see chapter Conversion data.

Example

```
SETVAL    5,op1        ; sets 5 to the op1 variable
SETVAL    2,op2        ; sets 2 to the op2 variable
MUL       op1,op2,var

;The value set in the var variable is 10
```

NOT

Syntax

NOT **operand**

Arguments

operand variable or name of device

Description

It performs a binary NOT operation (*the single bits are inverted*) on the value expressed by **operand**. The result is stored in **operand**.

Example

```
SETVAL    5,var        ; sets a value of 5 to "var"
NOT       var

 ; The result is var = -6
 ; Binary notation: 5 = 0000 0101,
 ; Binary notation:10 = 0000 1010
 ; Hexadeciaml notation  5 = 0000 0000 0000  0005
 ; Hexadeciaml notation 10 = 0000 0000 0000 000A
 ; by executing a NOT on value 5 the result is 0xFFFF FFFF FFFF FFFA = -6
```

OR

Syntax

OR **operand1, operand2, result**

Arguments

operand1	constant or variable or name of device
operand2	constant or variable or name of device
result	variable or name of device

Description

It carries out a binary OR operation (*between two bits, the result is 1 if at least one equals1*) between **operand1** and **operand2** and puts the result in **result**. To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
;The values set in the var variable is 7
;(Binary notation: 5 = 0101, 3 = 0011, 7 = 0111 )

OR        5,3,var
```

RANDOM

Syntax

RANDOM **min, max, result**

Arguments

min	constant or variable
max	constant or variable
result	variable or name of device

Description

It send to result a pseudocasual number included between **min** and **max** (extremes included). By executing the instruction repeatedly you obtain a sequence of pseudocasual numbers. To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
SETVAL    2,op1        ; sets 2 in the op1 variable
SETVAL    100,op2      ; sets 100 in the op2 variable
RANDOM    op1,op2,var

;The value set in the var variable is a random number
;included between 2 and 100
```

RESETBIT

Syntax

RESETBIT **mask, nbit**

Arguments

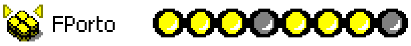
mask	constant or integer variable or countername or portname. It indicates the value to be modified (max 32 bit)
nbit	constant or integer variable or countername. Number of bit to be modified

Description

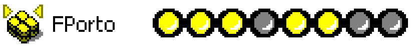
It sets a single bit of the passed bit **mask**, specified by **nbit**, to 0. The argument **mask** must correspond to an integer value with a maximum of 32 bit. The number of bits, **nbit**, ranges between 1 and 32.

Example

State of the port before executing the code



State of the port after executing the code



```
;---------------------------------------------------------
; Example to disable the line of a flag port:
;---------------------------------------------------------

SETVAL    2,nbit
RESETBIT  FlagPort,nbit

; disables line 2 of the flag port
```

ROUND

Syntax

ROUND **operand, result**

Arguments

operand	constant or variable or name of device
result	variable or name of device

Description

It performs a rounding operation on the **operand** and puts the value in **result**. To convert data, according to the type of declared data, see chapter Conversion data.

Example

```
SETVAL    5.7,op       ;sets 5.7 in the op variable
ROUND     op,var

;The value set in the var variable is 6

SETVAL    5.2,op       ;sets 5.2 in the op variable
ROUND     op,var
```

```
;The value set in the var variable is 5
```

SETBIT

Syntax

SETBIT	**mask, nbit**

Arguments

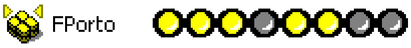
mask	constant or integer variable or countername or portname. Value to be modified (max 32 bit)
nbit	constant or integer variable or countername. Number of the bit to be modified (1÷32)

Description

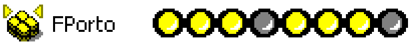
It sets a single bit of the passed bit **mask**, specified by **nbit**, to 1. The argument **mask** must correspond to an integer value with a maximum of 32 bit. The number of bits, **nbit**, ranges between 1 and 32.

Example

```
State of the port before code execution
```



```
State of the port after code execution
```



```
;---------------------------------------------------
; Example to enable a line of the flag port:
;
;---------------------------------------------------

SetVal      2,nbit
Setbit      FlagPort,nbit

; it enables line 2 of the flag port
```

SHIFTL

Syntax

SHIFTL	**operand 1 [, operand2]**

Arguments

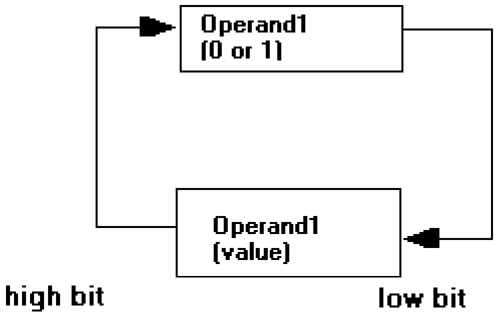
operand1	variable (integer or char) or name of device
operand2	variable (integer or char) or name of device

Description

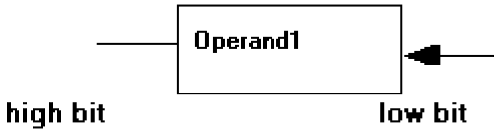
If **operand2 is** not specified, this instruction performs a left hand shift operation of the bits that make up the **operand1.** If also the second operand is specified, a rotation is performed between **operand2**, used as 0-value or not equal to 0 and the bits of **operand1**. In this case, at the end of the operation, **operand2,** will contain the carry of the operation and the bit of lower weight of **operand1** will become 0 or 1 according to the initial value of **operand2,** (0 or not equal to zero).

Example

Rotation (left hand shift with carry)



Example of left hand shift without carry



SHIFTR

Syntax

SHIFTR **operand1 [, operand2]**

Arguments

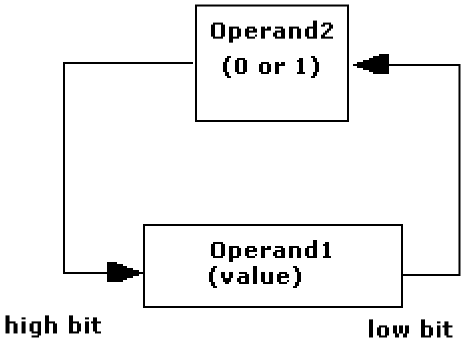
operand1	variable (integer or char) or name of device
operand2	variable or name of device

Description

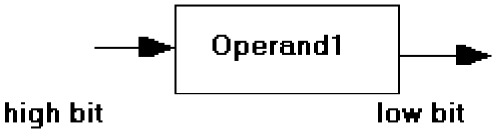
If **operand2** is not specified, this instruction performs a left - hand scrolling operation of the bits that make up the **operand1**. If also the second operand is specified, a rotation between **operand2,** used as 0-value or not equal to 0 and the bits of **operand1**, is performed. In this case, at the end of the operation **operand2** will contain the carry of the operation; the bit of lower weight of **operand1** will become 0 or 1 according to the initial value of **operand2** (0 or not equal to zero).

Example

Rotation (right-hand shift with carry)



Right hand shift (Right-hand shift without carry)



SIN

Syntax

SIN **operand, result**

Arguments

operand	constant or variable or name of device
result	variable or name of device

Description

It carries out a sinus operation on **operand** and puts the result in **result**.
The argument **operand** is expressed in degrees with a possible centesimal fractionary part (ex.: 30° 15" = 30,25.). To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
SetVal      30,op       ;sets 30 in the op variable
Sin         op,var

;The value set in the var variable is 0.5
```

SQR

Syntax

SQR **operand, result**

Arguments

operand	constant or variable or name of device
result	variable or name of device

Description

It extracts the square root of **operand** and puts the value in **result**.
Only positive values are admitted in the **operand** parameter. To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
SetVal      81,op       ;sets 81 in op variable
Sqr         op,var

;The value set in the var variable is 9
```

SUB

Syntax

SUB **operand1, operand2, result**

Arguments

operand1	constant or variable or name of device
operand2	constant or variable or name of device
results	variable or name of device

Description

It performs a subtraction operation between **operand1** and **operand2** and puts the result in **result.**
To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
SetVal      10,op1      ; sets 10 in the op1 variable
SetVal      4,op2       ; sets 4 in the op2 variable
Sub         op1,op2,var

;The values et in the var variable is 6
```

TAN

Syntax

TAN **operand, result**

Arguments

operand	constant or variable or name of device
result	variable or name of device

Description

It performs a tangent operation in **operand** and puts the result in **result**.
The **operand** argument is expressed in degrees. To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
SetVal      45,op       ;sets 45 in the op variable
Tan         op,var

;The value set in the var variable is 1
```

TRUNC

Syntax

TRUNC **operand, result**

Arguments

operand	constant or variable or name of device
result	variable or name of device

Description

It truncates to integer the value of **operand** and puts the result in **result**. (the decimal part goes lost). To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
SetVal      5.7,op      ;sets 5.7 to the op variable
Trunc       op,var

;The value set in the var variable is 5
```

XOR

Syntax

XOR	**operand1, operand2, result**

Arguments

operand1	constant or variable or name of device
operand2	constant or variable or name of device
result	variable or name of device

Description

It performs a binary XOR operation (between two bits, the result is one if only one of the two equals one) between **operand1** and **operand2** and puts the result in **result**. To convert data, according to the type of declared data, see chapter Data conversion.

Example

```
Xor   5,3,var

;The value set in the var variable is 6
;(Binary notation: 5 = 0101, 3 = 0011, 6 = 0110)
```

3.2.11 Multitasking

ENDMAIL

Syntax

ENDMAIL	**mail**

Arguments

mail	constant or integer variable. Number of post box (1÷256)

Description

It indicates the end of execution of a command associated to a message taken from the **mail** post box.
The task that sent the message (using the SENDMAIL instruction) and was waiting for command execution (wait arguments WAITACK) can now carry on with its own execution. This instruction ***is effective only when executed within task*** that previously received the message (with the WAITMAIL or TESTMAIL instruction).

See also instructions SENDMAIL, WAITMAIL and TESTMAIL

Example

Axis movement server

ENDREALTIMETASK

Syntax

ENDREALTIMETASK	**functionname**

Arguments

functionname	name of function

Description

It stops the execution of a real time task. See also STARTREALTIMETASK.

ENDTASK

Syntax

ENDTASK	**[taskname]**

Arguments

taskname	name of task

Description

It interrupts the execution of a task together with all the tasks activated by it (child tasks). This instruction also interrupts axis movement, cancels pending RECEIVEs and closes any connections with the serial ports.
If the **taskname** variable is omitted, it ends the execution of the current task.

GETPRIORITYLEVEL

Syntax

GETPRIORITYLEVEL	**level[,functionname]**

Arguments

level	variable. Execution priority level
functionname	name of function

Description

It returns the priority value of the task indicated by **functionname** to the **level** variable. This value is a number included between 1 and 255, where 1 indicates the highest priority level and 255 the lowest. If **functionname** is not specified, the priority value returned is the value of the current task, that is the function in which the GETPRIORITYLEVEL instruction is executed
See also SETPRIORITYLEVEL.

GETREALTIME

Syntax

GETREALTIME	**varname**

Arguments

varname	integer variable

Description

It returns to the **varname** variable the amount of time elapsed since the beginning of the last real-time axis handling. Time is expressed in microseconds. See also GETREALTIMECOUNT.

GETREALTIMECOUNT

Syntax

GETREALTIMECOUNT	**varname**

Arguments

varname	integer variable

Description

It returns to the **varname** variable the number of real-time axis-handlings executed since the last numeric control initialization. See also GETREALTIME.

HOLDTASK

Syntax

HOLDTASK	**[nametask]**

Arguments

nametask	name of task

Description

It interrupts the execution of the task defined in **nametask**. This instruction does not stop axis movement, which has to be interrupted through the STOP instruction.
If **nametask** is omitted, it interrupts the task in progress.

RESUMETASK

Syntax

RESUMETASK	**[nametask]**

Arguments

nametask	name of task

Description

It reactivates the execution of the task specified in **nametask**. If **nametask** is omitted, it reactivates the execution of the current task. If the task was interrupted using the STOPTASK instruction, axis movement is resumed as well.

SENDMAIL

Syntax

SENDMAIL	**mail, wait [, varname1 [,..varnameN]]**
SENDMAIL	**mail, wait, matrix[row]**

Arguments

mail	constant or integer variable. Mailbox number (1÷256)
wait	predefined constant. Command read or command execution wait mode. The values that can be attributed to the wait constant are: **WAIT** waits for the command to be read **NOWAIT** does not wait for the command to be read **WAITACK** waits for command execution
varname1[...varnameN]	constant or integer variable. Names of variables 1÷20
matrix[row]	constant or integer variable. Matrix row number

Description

It sends a message (or command) to the **mail** box. The messages can be used to synchronise and exchange information between two or more tasks.
If the **mail** box does not exist, meaning that no WAITMAIL or TESTMAIL instruction has been executed, the instruction is simply ignored.
If the receiver task is not waiting for a message (WAITMAIL instruction) or is engaged, the data sent from the instruction is saved in a queue. In this case:
1.if the wait argument is **NOWAIT,** execution carries on with the following instruction;
2.if the wait argument is **WAIT,** execution waits for the message to be read by the receiver task;
3.if the wait argument is **WAITACK,** execution waits for the message to be read and the execution of the command to be confirmed by the receiver task (through the same instruction or a new WAITMAIL).
It is very important that the number of the variables and their type coincide with those used to create the mail box with the WAITMAIL instruction. The control does not allow using different types and does not use automatic type conversion (cast) as usually happens.
A SENDMAIL without optional parameters (data) functions simply as a task synchronisation mechanism.

Example

Axis movement server

SETPRIORITYLEVEL

Syntax

SETPRIORITYLEVEL **level [, functionname]**

Arguments

level	constant or variable. Execution priority level.
functionname	name of function

Description

It sets in the **level** variable, the priority value of the task defined in **functionname**. This value is a number included between 0 and 255, where 0 indicates the highest priority level and 255 the lowest. If the name of the task is not specified in the **functionname** variable, it modifies the value of the current task, that is the execution level of the function in which the instruction is executed.

See also GETPRIORITYLEVEL.

STARTTASK

Syntax

STARTTASK **taskname [, parameters]**

Arguments

taskname	name of task
parameters	any parameters needed during task execution

Description

It activates the execution of the task defined in the **taskname**variable.
Any parameters needed during execution can be passed to the task. The number and type of the parameters must match the ones declared in the function implementing the task. If the task is already in execution the instruction does not have any effect.

Example

Parallel/Sequential execution

STARTREALTIMETASK

Syntax

STARTREALTIMETASK **functionname**

Arguments

functionname	name of function

Description

It activates the execution of a real time task. This kind of task is executed with the same frequency as the axis control real time. Unlike normal GPL tasks, every real time is executed entirely, from the first function instruction to the first FRET instruction. See also ENDREALTIMETASK.

Note:

The local variables declared in the realtime task are initialized only by the start of the task and then they maintain the value of the last run.

STOPTASK

Syntax

STOPTASK **taskname**

Arguments

taskname	name of task

Description

It stops the execution of a task and of all the tasks executed by it (child tasks), interrupting axis movement (if in progress).
If **taskname** is omitted, it stops execution of the current task. Task execution and axis movement can be reactivated through the RESUMETASK instruction.

WAITMAIL

Syntax

WAITMAIL	**mail [, varname1 [,..varnameN]]**
WAITMAIL	**mail, matrix[row]**

Arguments

mail	constant or integer variable. Mailbox number (1÷256)
varname1[...varnameN]	constant or integer variable. Names of variables 1÷20
matrix[row]	constant or integer variable. Matrix row number

Description

It receives a message from the **mail** mail box. The message may come with attached data.
The data received with the message is memorised in the indicated **varname** variables (1÷20) or in the matrix row specified by **matrix[row]**.
If no other messages are waiting to be read when the WAITMAIL instruction is executed, the task is put in HOLD state, which is terminated only when another task sends a message to the box with the SENDMAIL instruction.
The congruence between the old data and the data expected by the instruction, is checked during instruction execution.
A WAITMAIL without optional parameters is reduced to a simple synchronisation mechanism between tasks.
See also instructions SENDMAIL, ENDMAIL and TESTMAIL

Example

Axis movement server

WAITTASK

Syntax

WAITTASK	**taskname**

Arguments

taskname	name of task

Description

It waits for the **taskname** task to end execution.

Example

Sequential/Parallel execution

3.2.12 Flux management

CALL

Syntax

CALL	**subprogramname**

Arguments

subprogramname	name of subprogram, label

Description

It executes the subprogram specified by the **subprogramname** label.
Each subprogram, to return to the next CALL instruction, must end in the exit point with the instruction: RET.

Note

Together with RET, this instruction is a typical source of programming errors. We recommend taking great care when using it, in particular we suggest positioning the subprocedures at the end of the body of the function (after the FRET instruction) so as to avoid accidental execution of the subprocedure, as if it were an integral part of the main code. This situation, in the best of hypothesis, generates a system error; in other cases it causes anomalous behaviour of the machine whose origin is difficult to recognise.

FCALL

Syntax

[FCALL]	**functionname [, parameters]**
functionname	**[parameters]**

Arguments

functionname	name of the function to be called
parameters	any parameters passed to the function

Description

It calls a function, meaning that the **functionname** function is executed.
Any necessary **parameters** are passed to the function. These must match in number and type the parameters declared in the call function.
Execution of the caller function (the one where the FCALL is executed) restarts at the end of the call function (the one specified in the **functionname** parameter).

Note the difference from the STARTTASK instruction, which sends another function in execution in parallel with the caller function (it is used to have more tasks in execution at the same time).

Example

Sequential/Parallel execution

DELONFLAG

Syntax

DELONFLAG	**flagname**

Arguments

flagname	name of flag device

Description

It disables the software interruption management on the state of a flag bit or flag switch which was previously enabled with the ONFLAG instruction.

DELONINPUT

Syntax

DELONINPUT	**nameinput**

Arguments

nameinput	name of input

Description

It disables the software interruption management on the state of an input which was previously enabled with the ONINPUT instruction.

FOR/NEXT

Syntax

```
FOR                          index, begin, end [, step]
            instruction
            instruction
            ...
NEXT
```

Arguments

index	variable or countername
begin	constant or variable or countername. Beginning value
end	constant or variable or countername. End value
step	constant or variable or countername. Increase or decrease step

Description

It repeats cyclically the execution of the instructions included between the FOR instruction and the NEXT instruction.
During the first cycle the **index** variable is set on the value of the **begin** variable. In the second cycle the value of the **index** variable will equal (**begin**+**step**), and so on until the **index** variable is greater (or smaller, if the **step** variable is a negative value), than the **end** variable. If the **step** variable is omitted, a default value equal to +1 is set.
The instructions included between FOR and NEXT can modify the number of repetitions by modifying **index**.
When the repetitions end, it executes the instruction after NEXT.

Example

```
Function Loop
 local    i As integer
 local    vector[10] as integer

 For      i,1,10
    Setval      i, vector[i]      ; it fills in the elements
                                  ; of the vector
                                  ; with numbers 1,2, .... 10
 Next
 Fret



Function loop2
 local    j As integer
 local    vector[10] as integer

 For      j,1,10,2

    Setval      27, vector[j]     ; sets the value 27 in the following
                                  ; element of the vector: 1,3,5,7,9
 Next
 Fret
```

FRET

Syntax

FRET

Arguments

no argument

Description

Return from a function. It causes the interruption of the execution of a function and the release of the memory allocated for the local variables. If the function was sent in execution with an FCALL, caller function execution restarts from the next instruction.
If any WAITASKS were executed previously with the current function (the one in which the FRET is executed) as argument, the waiting tasks are released.

GOTO

Syntax

GOTO **label**

Arguments

label	label

Description

It makes an inconditional jump to the label specified in the **label** parameter.
A label is defined by a keyword followed immediately by the character ":".
The label must be contained in the body of the function in which the GOTO instruction is executed.

Note

The body of a function is the part included between the FUNCTION instruction, which declares the name of the function, and the instruction defining the following function (or the end of the file). It is clear, then, that it is possible to jump from the main body of the function to any existing subprocedures (see CALL and RET instructions). We highly discourage this programming style as it generates numerous errors which are difficult to identify.

Example

```
 ; Function to make a flag flash
 ; (for ex. a warning light on a synoptic panel)

Function Loop

loop:
 Setflag    alarm      ; enables the flag
 delay      1
 resetflag  alarm      ; disables the flag
 delay      1
 goto       loop
 Next
 Fret
```

IF/IFVALUE/IFTHENELSE

Syntax

```
IF                         varname, comparison operator, value, GOTO label
IF                         varname, comparison operator, value, CALL subprogramname
IF                         varname,comparison operator, value, functionname

IF                         varname, comparison operator, value THEN
          instruction
          instruction
          ...
ENDIF

IF                         varname, comparison operator, value THEN
          instruction
          instruction
          ...
ELSE
          instruction
          instruction
          ...
ENDIF
```

Arguments

varname	constant or variable or devicename
comparison operator	the symbols used for comparison are: **<** (smaller) **=** (equal) **>** (greater) **=<** (minor or equal) **>=** (greater or equal) **<>** (different)
value	constant or variable or devicename

label	name of the label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

The IF and IFVALUE instructions are synonimus. We suggest using the short version.
The instruction allows you to make a comparison between **varname** and **value** and, according to the result, to execute an action.
In the first three forms, if the comparison is positive, it can jump to label (GOTO), call a subprogram (CALL) or call a function (functionname). When the execution of the function or subprogram ends, it carries on from the following line. If the comparison is negative, the execution of the program continues. The IF...THEN construction allows to carry out one or more instructions conditionally. The instructions included between the keywords THEN and ENDIF are executed if the comparison between **varname** and **value** is positive.
The IF...THEN...ELSE construction allows you to define two blocks of instructions, of which only one will be executed. If the comparison between **varname** and **value** is positive, the instructions included between the keywords THEN and ELSE will be executed, if it's negative it will execute the instructions included between the words ELSE and ENDIF. In both cases the execution then continues with the instruction following ENDIF.

Note

IFVALUE is kept for compatibility with earlier GPL versions.

IFACC

Syntax

```
IFACC                axis, GOTO label
IFACC                axis, CALL subprogramname
IFACC                axis, functionname
```

Arguments

axis	name of axis device
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It checks whether the axis specified in the **axis** variable is in acceleration.
If it is, it jumps to **label** or calls **subprogramname** or **functionname**.

IFAND

Syntax

```
IFAND                operand1, operand2, testvalue, GOTO label
IFAND                operand1, operand2, testvalue, CALL subprogramname
IFAND                operand1, operand2, testvalue, functionname


IFAND                operand1, operand2, testvalue THEN
          instruction
          instruction
          ...
ENDIF

IFAND                operand1, operand2, testvalue THEN
          instruction
          instruction
          ...
ELSE
          instruction
          instruction
          ...
ENDIF
```

Arguments

operand1	constant or variable or devicename

operand2	constant or variable or devicename
testvalue	constant. Value used to check the result of the operation. Possible values are: **TRUE** 1 **FALSE** 0
label	name of the label to jump to
subprogramname	name of the subprogram
functionname	name of the function

Description

Two comparisons are performed, the first between **operand1** and **operand2**, the second between the result of the first comparison and **testvalue**.
The first comparison consists of a binary AND between **operand1** and **operand2**. The two operands are interpreted as bit masks. If in the result of the binary AND at least one bit is not equal to 0, the result of the first comparison is TRUE. This will then be compared with **testvalue**. If the two values coincide, a jump to label or a call function or call subprogram is performed.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

IFBIT

Syntax

```
IFBIT                   mask, nbit, state, GOTO label
IFBIT                   mask, nbit, state, CALL subprogramname
IFBIT                   mask, nbit, state, functionname


IFBIT                   mask, nbit, state THEN
          instruction
          instruction
          ...
ENDIF

IFBIT                   mask, nbit, state THEN
          instruction
          instruction
          ...
ELSE
          instruction
          instruction
          ...
ENDIF
```

Arguments

mask	constant or integer variable or countername or nameport. Value to be verified
nbit	constant or integer variable or countername. Number of the bit (1÷32)
state	predefined constant. State to be verified on mask. Acceptable values are: **ON** chosen bit to 1 **OFF** chosen bit to 0
label	jump label (GOTO)
subprogramname	call subprogram (CALL)
functionname	name of function

Description

Test on a single bit of the passed bit **mask**. The **mask** argument must correspond to an integer value with a maximum of 32 bits. The number assigned to the **nbit** variable to identify the bit to be tested must be included between 1 and 32. If the condition indicated in **state** is satisfied, it jumps to **label** or calls **subprogramname** or **functionname**.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

IFBLACKBOX

Syntax

```
IFBLACKBOX          GOTO label
IFBLACKBOX          CALL subprogramname
IFBLACKBOX          functionname
```

Arguments

label	name of the label to jump to
subprogramname	subprogram name
functionname	function name

Description

If the record is active, it jumps to **label** or it calls **subprogramname** or **functionname.** See also STARTBLACKBOX, PAUSEBLACKBOX and ENDBLACKBOX.

IFCHANGEVEL

Syntax

```
IFCHANGEVEL          axis [, state], GOTO label
IFCHANGEVEL          axis [, state], CALL subprogramname
IFCHANGEVEL          axis [, state], functionname
```

Arguments

axis	name of axis device
state	type of variation. Acceptable values are: **POSITIVE** **NEGATIVE**
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It tests if axis speed has varied.
If the axis specified in the **axis** variable is subject to speed variation during movement, a jump to **label** or a call to **subprogramname** of **functionname** is peformed.
The **state** parameter specifies if speed has increased (POSITIVE) or decreased (NEGATIVE).

IFCOUNTER

Syntax

```
IFCOUNTER          countername, comparison operator, value, GOTO label
IFCOUNTER          countername, comparison operator, value, CALL
                   subprogramname
IFCOUNTER          countername, comparison operator, value, functionname


IFCOUNTER          countername, comparison operator, value THEN
          instruction
          instruction
          ...
ENDIF

IFCOUNTER          countername, comparison operator, value THEN
          instruction
          instruction
          ...
ELSE
          instruction
          instruction
          ...
ENDIF
```

Arguments

countername	name of the counter
comparison operator	the symbols used for comparison are:

	< (smaller) **=** (equal) **>** (greater) **=<** (minor or equal) **>=** (greater or equal) **<>** (different)
value	constant or variable or countername
label	name of the label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

This instruction tests the counter.
If the content of the counter defined in the **countername** variable satisfies the condition specified by the **comparison operator,** with the value expressed in the **value** variable, it jumps to the label specified in **label** or calls the subprogram defined in **subprogramname** or the function defined in **functionname**.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

IFDEC

Syntax

```
IFDEC                     axis, GOTO label
IFDEC                     axis, CALL subprogramname
IFDEC                     axis, functionname
```

Arguments

axis	name of axis device
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It checks if the axis defined in the **axis** variable is decelerating.
If the condition is confirmed, it jumps to **label** or calls **subprogramname** or **functionname**.

IFDIR

Syntax

```
IFDIR                     axis, direction, GOTO label
IFDIR                     axis, direction, CALL subprogramname
IFDIR                     axis, direction, functionname


IFDIR                     axis, direction THEN
          instruction
          instruction
          ...
ENDIF

IFDIR                     axis, direction THEN
          instruction
          instruction
          ...
ELSE
          instruction
          instruction
          ...
ENDIF
```

Arguments

axis	name of axis device
direction	axis direction. Acceptable values are: **POSITIVE** positive axis direction **NEGATIVE** negative axis direction
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It tests the current direction of an axis.
If the **axis** is moving in the direction specified in the **direction** variable, a jump to **label** or a call to **subprogramname** or **functionname** is performed.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

IFERRAN

Syntax

```
IFERRAN                axis, comparison operator, value, GOTO label
IFERRAN                axis, comparison operator, value, CALL subprogramname
IFERRAN                axis, comparison operator, value, functionname


IFERRAN                axis, comparison operator, value THEN
         instruction
         instruction
         ...
ENDIF

IFERRAN                axis, comparison operator, value THEN
         instruction
         instruction
         ...
ELSE
         instruction
         instruction
         ...
ENDIF
```

Arguments

axis	name of axis device
comparison operator	the symbols used for comparison are: **<** (smaller) **=** (equal) **>** (greater) **=<** (minor or equal) **>=** (greater or equal) **<>** (different)
value	constant or variable or countername
label	name of the label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It checks the value of the tracking error (loop error) of the axis defined in the **axis** variable.
If the **axis** loop error confirms the condition expressed by the **comparison operator** with the value expressed by **value**, it jumps to **label** or calls **subprogramname** or **functionname**.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

IFERROR

Syntax

```
IFERROR               number, IDposiz, GOTO label
IFERROR               number, IDposiz, CALL label
IFERROR               number, IDposiz, functionname
IFERROR               devicename, state,IDposiz, GOTO label
IFERROR               devicename, state,IDposiz, CALL label
IFERROR               devicename, state,IDposiz, functionname
```

Arguments

number	DEFMSG or constant or integer variable
devicename	name of device
state	predefined constant. State to be verified Acceptable values are: **ON** **OFF**

IDposiz	constant or variable. A numeric value used in synoptics.
label	name of label to jump to
functionname	name of function

Description

It tests if cycle error is enabled.
If cycle error, identified by **number** and **IDposiz** or by **devicename, state** and **IDposiz,** is enabled it can jump to **label** or call function **functionname.**
Parameter **number** can identify an error of module cycle (therefore an entire numeric value) or of group (in this case a DEFMSG is used).
Parameter **devicename** is the name of a device and the parameter **state** represents the state ON/OFF in which the device is located, when the error is generated.
Parameter **number** can identify an error of module cycle (therefore an entire numeric value) or of group (in this case a DEFMSG is used).
Parameter **devicename** is the name of a device and the parameter **state** represents the ON/OFF state in which the device should be found, when the error is generated.
Parameter **IDposiz** is an optional parameter, specifying the numeric value used in the synoptics to sort out cicle errors in different cells. It must match the specified value in the synoptics creator for that particular display cell. If there is no need to point out a specific cell, the predefined NOPLACE constant must be assigned. The range of the values that can be set is included between 0 (NOPLACE) and 1023.
If the instruction is used without enabling the alarms management to status conditions, an error system is generated.
See also instruction ERROR.

IFFLAG

Syntax

```
IFFLAG                      flagname, state, GOTO label
IFFLAG                      flagname, state, CALL subprogramname
IFFLAG                      flagname, state, functionname


IFFLAG                      flagname, state THEN
          instruction
          instruction
          ...
ENDIF

IFFLAG                      flagname, state THEN
          instruction
          instruction
          ...
ELSE
          instruction
          instruction
          ...
ENDIF
```

Arguments

flagname	name of flag device
state	predefined constant. State to be tested. Possible values are: **ON** enabled **OFF** disabled
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It tests the logical state of a flag.
If the flag defined in the **flagname** variable satisfies the indicated **state**, it jumps to **label** or calls **subprogramname** or **functionname**.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

IFOR

Syntax

```
IFOR                    operand1, operand2, testvalue, GOTO label
IFOR                    operand1, operand2, testvalue, CALL subprogramname
IFOR                    operand1, operand2, testvalue, functionname


IFOR                    operand1, operand2, testvalue THEN
        instruction
        instruction
        ...
ENDIF

IFOR                    operand1, operand2, testvalue THEN
        instruction
        instruction
        ...
ELSE
        instruction
        instruction
        ...
ENDIF
```

Arguments

operand1	constant or variable or devicename
operand2	constant or variable or devicename
testvalue	constant. Value used to check the result of the operation. Possible values are: **TRUE** 1 **FALSE** 0
label	name of the label to jump to
subprogramname	name of the subprogram
functionname	name of the function

Description

Two comparisons are performed, the first between **operand1** and **operand2**, the second between the result of the first comparison and **testvalue**.
The first comparison consists of a binary OR between **operand1** and **operand2**. The two operands are interpreted as bit masks. If in the result of the binary OR at least one bit is not equal to 0, the result of the first comparison is TRUE. This will then be compared to **testvalue**. If the two values coincide, a jump to label or a call function or call subprogram is performed.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

IFINPUT

Syntax

```
IFINPUT                 inputname, state, GOTO label
IFINPUT                 inputname, state, CALL subprogramname
IFINPUT                 inputname, state, functionname


IFINPUT                 inputname, state THEN
        instruction
        instruction
        ...
ENDIF

IFINPUT                 inputname, state THEN
        instruction
        instruction
        ...
ELSE
        instruction
        instruction
        ...
```

```
ENDIF
```

Arguments

inputname	name of input
state	predefined constant. State to be verified Acceptable values are: **ON** enabled **OFF** disabled
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It tests the analog state of an input.
If the input specified in the **inputname** variable is in the indicated **state**, a jump to **label** or a **subprogramname** or **functionname** call is performed.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

IFMESSAGE

Syntax

```
IFMESSAGE          number, IDposiz, GOTO label
IFMESSAGE          number, IDposiz, CALL label
IFMESSAGE          number, IDposiz, functionname
```

Arguments

number	DEFMSG or constant or integer variable
IDposiz	constant or variable. A numeric value used in synoptics.
label	name of label to jump to
functionname	name of function

Description

It tests if message is enabled.
If message, identified by **number** and **IDposiz** is enabled it can jump to **label** or call function **functionname,**
Parameter **IDposiz** is an optional parameter specifying the numeric value used in synoptics to sort out cycle errors in different cells. It must correspond with the specified value in the synoptics creator for that particular display cell. If there is no need to point out a specific cell, the predefined NOPLACE constant must be assigned. The range of the values, that can be set is included between 0 (NOPLACE) and 1023.
If the instruction is used without enabling the alarms management to status conditions, an error system is generated.
See also instruction MESSAGE.

IFOUTPUT

Syntax

```
IFOUTPUT           outputname, state, GOTO label
IFOUTPUT           outputname, state, CALL subprogramname
IFOUTPUT           outputname, state, functionname


IFOUTPUT           outputname, state THEN
        instruction
        instruction
        ...
ENDIF

IFOUTPUT           outputname, state THEN
        instruction
        instruction
        ...
ELSE
        instruction
        instruction
```

```
            ...
ENDIF
```

Arguments

outputname	name of output
state	predefined constant. State to be verified on output Acceptable values are: **ON** enabled **OFF** disabled
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It tests the analog state of an output.
If the input specified in the **outputname** variable is in the indicated **state**, a jump to **label** or a **subprogramname** or **functionname** call is performed.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

IFQUOTER

Syntax

```
IFQUOTER            axis, comparison operator, value, GOTO label
IFQUOTER            axis, comparison operator, value, CALL subprogramname
IFQUOTER            axis, comparison operator, value, functionname


IFQUOTER            axis, comparison operator, value THEN
          instruction
          instruction
          ...
ENDIF

IFQUOTER            axis, comparison operator, value THEN
          instruction
          instruction
          ...
ELSE
          instruction
          instruction
          ...
ENDIF
```

Arguments

axis	name of axis device
comparison operator	the symbols used for comparison are: **<** (smaller) **=** (equal) **>** (greater) **=<** (minor or equal) **>=** (greater or equal) **<>** (different)
value	constant or variable or countername
label	name of the label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It tests the real position specified by the **axis** variable.
If the value of the **axis** variable complies with the condition expressed in the **comparison operator** with the value specified by **value**, it jumps to **label** or calls **subprogramname** or **functionname**.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

IFQUOTET

Syntax

```
IFQUOTET            axis, comparison operator, value, GOTO label
IFQUOTET            axis, comparison operator, value, CALL subprogramname
IFQUOTET                 axis, comparison operator, value, functionname


IFQUOTET                 axis, comparison operator, value THEN
           instruction
           instruction
           ...
ENDIF

IFQUOTET                 axis, comparison operator, value THEN
           instruction
           instruction
           ...
ELSE
           instruction
           instruction
           ...
ENDIF
```

Arguments

axis	name of axis device
comparison operator	the symbols used for comparison are: **<** (smaller) **=** (equal) **>** (greater) **=<** (minor or equal) **>=** (greater or equal) **<>** (different)
value	constant or variable or countername
label	name of the label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It tests the theoretical position specified by the **axis** variable.
If the value of the **axis** variable complies with the condition expressed in the **comparison operator** with the value specified by **value**, it jumps to **label** or calls **subprogramname** or **functionname**.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

IFRECEIVED

Syntax

```
IFRECEIVED               [source, ] identifier, GOTO label
IFRECEIVED               [source, ] identifier, CALL subprogramname
IFRECEIVED               [source, ] identifier, functionname
```

Arguments

source	string constant
identifier	string constant
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It tests if a RECEIVE instruction has been satisfied.
If a specified former RECEIVE was satisfied, it jumps to label or calls subprogramname or functionname.
See also instructions RECEIVE, WAITRECEIVE, SEND.

IFREG

Syntax

```
IFREG                    axis, GOTO label
IFREG                    axis, CALL subprogramname
IFREG                    axis, functionname
```

Arguments

axis	name of axis device
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It checks that the axis specified in the **axis** variable is in regime state.
If the condition is confirmed, it jumps to **label** or calls **subprogramname** or **functionname.**

IFSAME

Syntax

```
IFSAME                      operand1, operand2, GOTO label
IFSAME                      operand1, operand2, CALL subprogramname
IFSAME                      operand1, operand2, functionname
```

Arguments

operand1	variable or devicename
operand2	variable or devicename
label	name of the label to jump to
subprogramname	name of the subprogram
functionname	name of the function

Description

Test between two operands.
It verifies if the value defined in **operand1** and **operand2** refer either to the same device or the same memory area.
If the test between the two operands is confirmed, it jumps to **label** or calls **subprogramname** or **functionname.**

IFSTILL

Syntax

```
IFSTILL                     axis, GOTO label
IFSTILL                     axis, CALL subprogramname
IFSTILL                     axis, functionname
```

Arguments

axis	name of axis device
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It tests if the axis defined in the **axis** variable is really still, that is if it is "in position".
If the condition is confirmed, it jumps to **label** or calls **subprogramname** or **functionname.**
See also IFTARGET and IFWIN.

IFSTR

Syntax

```
IFSTR                       string1, comparison operator, string2, GOTO label
IFSTR                       string1, comparison operator, string2, CALL subprogramname
IFSTR                       string1, comparison operator, string2, functionname


IFSTR                       string1, comparison operator, string2 THEN
            instruction
            instruction
            ...
ENDIF

IFSTR                       string1, comparison operator, string2 THEN
            instruction
            instruction
            ...
```

```
ELSE
        instruction
        instruction
        ...
ENDIF
```

Arguments

string1	string variable. The first ASCII string
comparison operator	the symbols used for comparison are: **<** (smaller) **=** (equal) **>** (greater) **=<** (minor or equal) **>=** (greater or equal) **<>** (different)
string2	string variable. The second ASCII string
label	name of the label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

Test on ASCII strings.
If the string defined in **string1** confirms the condition expressed by the **comparison operator** with the string in **string2**, a jump to **label** or a **subprogramname** or **functionname** call is performed.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

IFTARGET

Syntax

```
IFTARGET        axis, GOTO label
IFTARGET        axis, CALL subprogramname
IFTARGET        axis, functionname
```

Arguments

axis	name of axis device
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It checks if the axis defined in the **axis** variable has reached the final programmed position. Even if it has reached the final target position, this does not necessarily mean that it has stopped, as it usually has to recover the loop error. If the condition is confirmed, it jumps to **label** or calls **subprogramname** or **functionname**.
See also IFSTILL and IFWIN.

IFTASKHOLD

Syntax

```
IFTASKHOLD      nametask, GOTO label
IFTASKHOLD      nametask, CALL subprogramname
IFTASKHOLD      nametask, functionname
```

Arguments

nametask	name of parallel task
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It checks whether the task has been interrupted (hold state).
If the **nametask** is in hold, a jump to **label** or a **subprogramname** or **functionname** call is performed.

IFTASKRUN

Syntax

```
IFTASKRUN            nametask, GOTO label
IFTASKRUN            nametask, CALL subprogramname
IFTASKRUN            nametask, functionname
```

Arguments

nametask	name of parallel task
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It checks if the task is in execution.
If the task defined in **nametask** is in execution, it jumps to **label** or calls **subprogramname** or **functionname**.

IFTIMER

Syntax

```
IFTIMER              nametimer, comparison operator, value, GOTO label
IFTIMER              nametimer, comparison operator, value, CALL
                     subprogramname
IFTIMER              nametimer, comparison operator, value, functionname


IFTIMER              nametimer, comparison operator, value THEN
          instruction
          instruction
          ...
ENDIF

IFTIMER              nametimer, comparison operator, value THEN
          instruction
          instruction
          ...
ELSE
          instruction
          instruction
          ...
ENDIF
```

Arguments

nametimer	name of timer device
comparison operator	the symbols used for comparison are: **<** (smaller) **=** (equal) **>** (greater) **=<** (minor or equal) **>=** (greater or equal) **<>** (different)
value	constant or variable or nametimer. The comparison value.
label	name of the label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

Timer test.
If the content of the **nametimer** timer satisfies the condition expressed in the **comparison operator** with the value expressed in value, a jump to **label** or a **subprogramname** or **functionname** call is performed.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

IFVEL

Syntax

```
IFVEL                axis, comparison operator, value, GOTO label
IFVEL                axis, comparison operator, value, CALL subprogramname
IFVEL                      axis, comparison operator, value, functionname


IFVEL                      axis, comparison operator, value THEN
            instruction
            instruction
            ...
ENDIF

IFVEL                      axis, comparison operator, value THEN
            instruction
            instruction
            ...
ELSE
            instruction
            instruction
            ...
ENDIF
```

Arguments

axis	name of axis device
comparison operator	the symbols used for comparison are: **<** (smaller) **=** (equal) **>** (greater) **=<** (minor or equal) **>=** (greater or equal) **<>** (different)
label	name of the label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It tests current speed of an axis.
If the speed of the axis confirms the condition expressed in the **comparison operator** with the value expressed in **valu**e, a jump to **label** or a **subprogramname** or **functionname** call is performed.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

IFWIN

Syntax

```
IFWIN                      axis, GOTO label
IFWIN                      axis, CALL subprogramname
IFWIN                      axis, functionname
```

Arguments

axis	name of axis device
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It tests if the axis specified in the **axis** variable has entered the arrival position window (see Glossary).
If the condition is confirmed, it jumps to **label** or calls **subprogramname** or **functionname**.
See also IFTARGET and IFSTILL.

IFXOR

Syntax

```
IFXOR                      operand1, operand2, testvalue, GOTO label
IFXOR                      operand1, operand2, testvalue, CALL subprogramname
IFXOR                      operand1, operand2, testvalue, functionname


IFXOR                      operand1, operand2, testvalue THEN
            instruction
```

```
                instruction
                ...
    ENDIF

    IFXOR                       operand1, operand2, testvalue THEN
                instruction
                instruction
                ...
    ELSE
                instruction
                instruction
                ...
    ENDIF
```

Arguments

operand1	constant or variable or devicename
operand2	constant or variable or devicename
testvalue	constant. Value used to verify the result of the operation. Possible values are: **TRUE 1** **FALSE 0**
label	name of the label to jump to
subprogramname	name of the subprogram
functionname	name of the function

Description

Two comparisons are performed, the first between **operand1** and **operand2**, the second between the result of the first comparison and **testvalue**.
The first comparison consists in a binary XOR between **operand1** and **operand2**. The two operands are interpreted as bit masks. If in the result of the binary XOR at least one bit is not equal to 0, the result of the first comparison is TRUE. This will then be compared to **testvalue**. If the two values coincide, a jump to label or a call function or call subprogram is performed.
For more details on the IF-THEN-ELSE construct, see IF / IFVALUE / IF-THEN-ELSE.

ONERRSYS

Syntax

```
    ONERRSYS                    functionname
```

Arguments

functionname	name of function

Description

It enables system error management. The normal behaviour of the control, when a system error occurs, is to interrupt all the tasks. The system error management allows you to avoid closing down the tasks for which it has been enabled.
When a system error occurs the **functionname** function is sent in execution. The function's task is to analyse the system error and carry out the necessary actions to secure the machine.

The **functionname** function has two limitations:
First of all, it has to accept the following parameters:

- the number of the system error, as Integer number.
- the task where the error took place, as Function
- the device which generated the error, as device.

Secondly, it can not contain certain GPL instructions. See List of instructions which can not be used with interrupt.

In the case of multiple System Errors, the function is called once for each error generated, in sequence. If the function itself generates a System Error, all tasks are interrupted.

During function execution, the task for which the error management has been enabled stops, and it only restarts at the end of the first function called by the ONERRSYS instruction.

Example

Main Cycle with error management

ONFLAG

Syntax

ONFLAG **flagname, [state,] functionname[,arguments]**

Arguments

flagname	name of flag device
state	predefined constant. State to be tested. Possible values are: **ON** enabled **OFF** disabled
functionname	name of function
arguments	any arguments of the function

Description

It enables software interruption of the task in which it is executed, according to the state of the flag specified. When the flag switches to the indicated state (interrupt), task execution is interrupted and the function specified in **functionname** is executed. At the end of this execution the task restarts from where it was interrupted. The function executed after the interrupt has certain limitations. Namely, not all GPL instructions can appear in the body of the function. This limitation is necessary to avoid critical interruptions of the GPL code or long waits. See List of instructions which can not be used with interrupt.
If the **state** argument is omitted, the function is called each time the state of the flag changes.
The test on the flag state is executed every 5ms, which means that maximum latency time, between flag variation and execution of the function, is 5 ms.
Only one ONFLAG can be defined on the same flag.
Vectors or local matrixes can not be arguments of the function defined in **functionname**.
See also instructions DELONFLAG, ONINPUT, DELONINPUT.

ONINPUT

Syntax

ONINPUT **nameinput, [state,] functionname [,arguments]**

Arguments

nameinput	name of input
state	predefined constant. State to be tested. Possible values are: **ON** enabled **OFF** disabled
functionname	name of function
arguments	any arguments of the function

Description

It enables software interruption of the task in which it is executed, according to the state of the input specified. When the input switches to the indicated state (interrupt), task execution is interrupted and the function specified in **functionname** is executed. At the end of this execution the task restarts from where it was interrupted. The function executed after the interrupt has certain limitations. Namely, not all GPL instructions can appear in the body of the function. This limitation is necessary to avoid critical interruptions of the GPL code or long waits. See List of instructions which can not be used with interrupt.
If the **state** argument is omitted, the function is called each time the state of the input changes.
The test on the input state is executed every 5ms, to which 4ms of anti-rebound filter on input management must be added. This means that latency time can reach 9 ms, before launching the function.
Only one ONINPUT can be defined on the same input.
See also instructions DELONINPUT, ONFLAG and DELONFLAG.

REPEAT/ENDREP

Syntax

```
REPEAT              value
        instruction
        instruction
```

```
                ...
    ENDREP
```

Arguments

value constant or variable or countername. Number of repetitions.

Description

It repeats the execution of the instructions enclosed between the REPEAT instruction and the ENDREP instruction as many times as indicated in the **value** variable.
When the program reaches the ENDREP instruction, the counter of the number of repetition decreases and, if its value is not less or equal to zero, the block of instructions is reexecuted starting from the instruction after REPEAT. This means that the instructions are executed at least once (even if the value parameter is naught or negative from the beginning).
When the repetitions are concluded, the instruction following ENDREP is executed.
See also instruction FOR/NEXT.

Example

```
; example of cycle moving an axis
; between two positions for 10 times
Function Cycleo
    Repeat   10
      MovAbs       axis,100
      waitinput    switch,ON
      Movabs       axis,-100
      Waitinput    switch, OFF
    EndRep
Fret
```

RET

Syntax

RET

Arguments

no arguments

Description

It ends the execution of a subprogram and returns to the instruction immediately after the call CALL.
See also the instruction CALL.

Note

This instruction, together with CALL, is a typical source of programming errors. We recommend taking great care when using it, in particular we suggest positioning the subprocedures at the end of the body of the function (after the FRET instruction) so as to avoid accidental execution of the subprocedure, as if it were an integral part of the main code. This situation, in the best of hypothesis, generates a system error; in other cases it causes anomalous behaviour of the machine whose origin is difficult to recognise.

SELECT

Syntax

```
SELECT varname

            CASE value
            GOTO label

            CASE value1 TO value2
            CALL subprogramname

            CASE IS  < = >  value
            [FCALL] functionname [parameter1,…parameterN]

            CASE ELSE
            GOTO label
```

ENDSELECT

Arguments

varname	constant or integer variable or countername
value, value1, value2	integer constants
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function
parameter1...parameterN	parameter passed to the call function

Description

Multiple selection with jump to **label**, call to **subprogramname** or **functionname** according to the **value** of the **varname** variable.
Each CASE (optional) can have only one GOTO, CALL or FCALL instruction.
At least one case must be included between SELECT and ENDSELECT. The latter indicates the end of the SELECT instruction.
After each CALL or FCALL the execution of the function continues with the instruction following ENDSELECT.
The CASE-ELSE branch is executed if no previous CASE is satisfied.

Example

Axis movement server

TESTIPC

Syntax

TESTIPC	**IPCname, [, varname1 [, varnameN, ...]], GOTO label**
TESTIPC	**IPCname, [, varname1 [, varnameN, ...]], CALL subprogramname**
TESTIPC	**IPCname, [, varname1 [, varnameN, ...]], functionname**
TESTIPC	**IPCname, matrix[row], GOTO label**
TESTIPC	**IPCname, matrix[row], CALL subprogramname**
TESTIPC	**IPCname, matrix[row], functionname**
TESTIPC	**IPCname, vector, GOTO label**
TESTIPC	**IPCname, vector, CALL subprogramname**
TESTIPC	**IPCname, vector, functionname**

Arguments

IPCname	string constant. Name of IPC
varname1[...varnameN]	constant or variable. Names ranging from1÷N
matrix[row]	constant or integer variable. Row number of matrix
vector	name of vector
matrix	name of matrix
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It tests and receives IPC commands.
When the TESTIPC instruction is executed for the first time the shared memory is allocated; the memory's dimension is calculated on the basis of the size of sent data. The maximum shared memory dimension is 64 Kb.
A semaphore is connected to the memory to allow the synchronisation of the tasks accessing it. The task accessing it checks if an active semaphore is present, it reads the data of the shared memory and disables the semaphore. Immediately after, a jump to label instruction, or the function or the program described as last parameter of the TESTIPC instruction, is executed.
See also SENDIPC and WAITIPC.

TESTMAIL

Syntax

TESTMAIL	**mail, [varname1 [,..varnameN]], GOTO label**
TESTMAIL	**mail, [varname1 [,..varnameN]], CALL label**

TESTMAIL	**mail, [varname1 [,..varnameN]], functionname**
TESTMAIL	**mail, matrix[row], GOTO label**
TESTMAIL	**mail, matrix[row], CALL subprogramname**
TESTMAIL	**mail, matrix[row], functionname**

Arguments

mail	constant or integer variable (1÷256). Number of mailbox
varname1[...varnameN]	integer variable. Names ranging from1÷20
matrix[row]	constant or integer variable. Row number of matrix
label	name of label to jump to
subprogramname	name of subprogram
functionname	name of function

Description

It tests and receives messages.
The first TESTMAIL in the **mail** mailbox creates the mailbox.
If the message is in the **mail** mailbox, the data sent with the message is saved either in the **varname** variables (1÷20), if they are are indicated, or in the row of the matrix indicated by **matrix[row]**; moreover it jumps to **label** or calls **subprogramname** or **functionname**.
During execution, congruence between passed data and expected date is verified.
See also instructions SENDMAIL, WAITMAIL and ENDMAIL.

3.2.13 Various

CLEARERRORS

Syntax

CLEARERRORS	**[IDposiz]**

Arguments

IDposiz	constant or variable. A numeric value used by synoptics

Description

It tells the supervisor PC to delete all the cycle errors concerning the module which is executing the instruction, previously sent by the ERROR instruction. The **IDposiz** parameter is an optional parameter that specifies the numeric value used in synoptics to sort cycle errors in different cells. It must match the value specified in the synoptic creator for that specific display cell. Albatros uses this identifier to manage cycle errors in separate queues. A new queue is created for each IDposiz. The range of the values that can be set is included between 0 (NOPLACE) and 1023. If the **IDposiz** parameter is not specified, all the cycle errors both in the default queue and in the possible other queues are deleted.
See also instructions ERROR and DELERROR

CLEARMESSAGES

Syntax

CLEARMESSAGES	**[IDposiz]**

Arguments

IDposiz	constant or variable. A numeric value used by synoptics

Description

It tells the supervisor PC to delete all the messages concerning the module which is executing the instruction, previously sent by the MESSAGE instruction. The **IDposiz** parameter is an optional parameter that specifies the numeric value used in synoptics to sort messages in different cells. It must match the value specified in the synoptic creator for that specific display cell. Albatros uses this identifier to manage messages in separate queues. A new queue is created for each IDposiz.The range of the values that can be set is included between 0 (NOPLACE) and 1023. If the **IDposiz** parameter is not specified, all the messages both in the default queue and in the possible other queues are deleted.
See also instructions MESSAGE and DELMESSAGE.

DEFMSG

Syntax

DEFMSG **label [, languageprefix1], "messagestring"**
, … ,
[, languageprefixN, "messagestring"]

Arguments

label mnemonic name of message to be displayed

languageprefix predefined constant. Language in which the message is written
Allowed values are:
ITA Italian
ENG English
FRA French
ESP Spanish
DEU German

messagestring message to be displayed. It must be written in inverted commas ("")

Description

It assigns a **label** to a message. The DEFMSG instruction must be declared before implementing the functions. The definition of the message can only be used inside the file (or group) in which it is declared. It is possible to insert messages in various languages by using the predefined constant **languageprefix**. In this case the MESSAGE instruction will display the message in the language currently used by Albatros. A message that no prefix is associated with is used when the language currently in use does not match any of the existing prefixes. A maximum of six messages can be defined per label: five with language prefix plus one with no language prefix.
All the labels of different languages can be written on the same line or on more lines, beginning a new paragraph each time by pressing the character "-" preceded by one space.
DEFMSG instruction can be passed as parameter to a function. In this way the function that recives it can use it as one of the three arguments of ERROR an MESSAGE. (See example 2).
See also instructions MESSAGE, DELMESSAGE, ERROR, DELERROR.

Example 1

```
;assigning a message string to a label without language selection
DEFMSG    MSG_GRU_1   "Message group 1"

;using the definition
MESSAGE   MSG_GRU_1         ;display: "Message group 1"

;assigning a message string to a label with language selection
DEFMSG    MSG_GRU_1   ITA "Messaggio gruppo 1"
                ENG "Message group 1"

;using the definition when Albatros is set on ENG
MESSAGE   MSG_GRU_1         ;display: "Message group 1"
```

Example 2:

```
Into a group:
DEFMSG  MSG_TEST  "Execution error"

FUNCTION ChiamaTest
        Test MSG_TEST
FRET

 Into a library:
    DEFMSG  MSG_BASE   "Error signal: $1"
    ...
FUNCTION Test Public
    PARAM  codice AS integer
    ERROR  MSG_BASE NOPLACE NOSTORE codice
FRET
Visualized cycle error is: Error signal: Execution error
```

DELAY

Syntax

DELAY	**value**

Arguments

value	constant or variable. Delay expressed in seconds.

Description

It waits as long as indicated in **value**. When time is up, the following instruction is executed. The programmable minimum value is 4 msec. (0,004 seconds)

DELERROR

Syntax

DELERROR	**devicename [,state [, IDposiz [STORE]]**
DELERROR	**number [, IDposiz [STORE]]**

Arguments

devicename	devicename
number	DEFMSG or constant or variable
IDposiz	constant or variable. A numeric value used in synoptics
state	predefined constant Possible values are: **ON** **OFF**
STORE	predefined numeric constant which, if enabled, allows you to save the cycle error in the current month's error report file

Description

It tells the supervisor PC to supervise the deletion of a cycle error previously sent by the ERROR instruction.
If the name of a device is specified, instead of the number, it sends the PC the type and logic address of the device. For the clearance to be effective, all the values set in the parameters must coincide with those used to generate the error. The STORE parameter, however, behaves anomalously. Errors saved on the report file are not eliminated from the file, but only from the error window, while a new registration of the error deletion is added to the file (the size of the file increases!).
Parameter **IDposiz** is an optional parameter, specifying the numeric value used in the synoptics to sort out cicle errors in different cells. It must match the specified value in the synoptics creator for that particular display cell. If there is no need to point out a specific cell, the predefined NOPLACE constant must be assigned. The range of the values that can be set is included between 0 (NOPLACE) and 1023.
If cycle errors are managed like warning signals, all cancel requests are sent. If alarms are managed like statuses, cycle error cancelling is only sent if cycle error is active, otherwise DELERROR instruction is ignored. See also instructions ERROR, CLEARERRORS.

DELMESSAGE

Syntax

DELMESSAGE	**number [, IDposiz]**

Arguments

number	DEFMSG or constant or variable
IDposiz	constant or variable. Numeric value used in synoptics

Description

It sends to the PC a request to delete a message previously sent with a MESSAGE instruction. If messages are managed like warning signals, all corresponding messages are cancelled. If messages are managed like statuses, message cancelling is only sent if its status was active, otherwise DELMESSAGE instruction is ignored.
Parameter **IDposiz** is an optional parameter, specifying the numeric value used in the synoptics to sort out cicle errors in different cells. It must match the specified value in the synoptics creator for that particular display cell. If there is no need to point out a specific cell, the predefined NOPLACE constant must be assigned. The range of the values that can be set is included between 0 (NOPLACE) and 1023.

See also instruction MESSAGE.

ERROR

Syntax

ERROR **devicename [,state [, IDposiz [, log]]]**
ERROR **number [, IDposiz [, log [, arg1, ..., arg3]]]**

Arguments

devicename	name of device
number	DEFMSG or constant or variable
IDposiz	constant or variable. A numeric value used in synoptics.
state	predefined constant Possible values are: **ON** **OFF**
log	predefined numeric constant Possible values are: **STORE** error saved to file **NOSTORE** error not saved to file
arg1, ..., arg3	constant or device or variable.

Description

It generates a cycle error. The error is identified by the **number** parameter or by the name of the device. The parameter **number** can identify a module cycle error (i.e. a whole number) or group cycle error (in this case, DEFMSG applies).
If the name of a device is specified, instead of the number, it sends the PC the type and logic address of the device. The cycle error is sent to the supervisor PC and displayed on the Albatros error bar.
The **IDposiz** parameter is used in synoptic views to sort cycle errors in different cells. It must match the value specified in the synoptic creator for that specific display cell. Albatros uses this identifier to manage cycle errors in separate queues. A new queue is created for each IDposiz. If the IDposiz parameter is not specified or when the predefined constant NOPLACE is used, the cycle error is located in the default queue with the value IDposiz=0.The range of the values that can be set is included between 0 (NOPLACE) and 1023.
Setting **log** parameter to **STORE** causes the cycle error to be saved in the error report file of the current month. A high number of generated or cleared errors may put the performance level of the remote modules at risks. In fact, the PC supervisor must control all the errors sent (and they possible clearance). This may slow down the sending of important data to the control, particularly the processing programs.
The optional **arg1, ..., arg3** parameters are used to define parameter error messages. The error message's definition string will feature markers that will be replaced - when the error is generated - with the value or name of the device or variable passed as a parameter. Markers to be inserted in the string are as follows:

- $1, ... $2 replaced with the ***name*** of the device or variable ($1 stands for arg1 etc.)
- $(1), ..., $(3) replaced with the ***value*** of the device or variable.

Types of data valid for the arg1, ..., arg3 parameters are as follows:

- CHAR
- INTEGER
- FLOAT
- DOUBLE (though it is automatically converted into FLOAT)
- message number (or DEFMSG label)
- device
- global or local variable
- function parameter. It can be used as function parameter the label defined by the DEFMSG instruction.

Strings, matrices and vectors cannot be used as parameters (although individual vector or matrix elements are valid). For local variables, only the value can be decoded, not the name.
For the purpose of deleting a message with the DELERROR instruction, the arg1, ...arg3 parameters are disregarded.

Two error management modes are defined and established by manufacturer of the machine:
Alarms managed like warning signals: all cycle errors are sent. Albatros keeps a queue of the last 10 errors of the specified queue and the last 100 errors of the default queue.
Alarms managed like statuses: error is considered active or inactive. If active, any further

sending of the same cycle error (by ERROR instruction) is ignored.
See also instructions DELERROR, CLEARERRORScannot be used.

Example 1

```
DEFMSG    ERR_TOOL    "Tool missing"
DEFMSG    ERR_TOOL_P  "Load tool $(1) in slot $(2)"

; tag for synoptic views
CONST           TOOLCHANGE = 5

; error shown in the Errors Bar or in not tagged sinoptic views' cells
ERROR           ERR_TOOL

; error saved in report file and shown in synoptic views' cells tagged
; with code 5
ERROR           ERR_TOOL, TOOLCHANGE, STORE

; error saved in report file but not dispatched to tagged synoptic
; views'cell
ERROR           ERR_TOOL, NOPLACE, STORE

; error with parameters
ERROR           ERR_TOOL_P, NOPLACE, NOSTORE, MxTools[3].Cod, 5
```

Example 2

```
    ; defined in a group
    DEFMSG      MSG_ERR_CARICO    "Error on loading tool"


    Function ShowError
        MsgTool  MSG_ERR_CARICO MxUtensili[3].Cod
    fret

    ; defined in a library
    DEFMSG      MSG_ERR_TOOL      "Error tool: $1 $(2)"


    Function MsgTool public
    PARAM parameter1 as integer
    PARAM parameter2 as integer

           MESSAGE MSG_ERR_TOOL NOPLACE parameter1 parameter2
    fret
```

IFDEF/ELSEDEF/ENDDEF

Syntax

```
IFDEF                   constant
          instruction
          ...
ENDDEF

IFDEF                   constant, comparison operator, value
          instruction
          ...
ENDDEF

IFDEF                   EXIST, namegroup
          instruction
          ...
ENDDEF
```

```
IFDEF                  LINKED, devicename
        instruction
        ...
ENDDEF

IFDEF                  UNLINKED, devicename
        instruction
        ...
ENDDEF

IFDEF                  constant, comparison operator, value
        instruction
        ...
ELSEDEF
        instruction
        ...
ENDDEF
```

Arguments

constant	integer, char, double, string constant
varname	integer, char, double or string constant
comparison operator	the symbols used for comparison are: **<** (smaller) **=** (equal) **>** (greater) **=<** (minor or equal) **>=** (greater or equal) **<>** (different)
value	constant or name of device
namegroup	name constant or name of group
devicename	string constant or device name

Description

The conditional compilation allows you to check which parts of a GPL function file must be compiled and executed. The compiler verifies that the condition requested as argument of the IFDEF instruction is satisfied. In this case it compiles the code included between the IFDEF instruction and the ENDDEF or ELSEDEF instruction. If an ELSEDEF instruction exists, and the condition is not satisfied, it will compile the code included between the ELSEDEF instruction and the ENDDEF instruction.
The compilation condition can be expressed in some different ways:

- a constant is specified after the IFDEF instruction. In this case the condition is satisfied if a global constant or a constant of the existing group with the specified name exists.
- A relation between two operators and an operand is specified after the IFDEF instruction. The first operand must be a constant. In this case the condition is satisfied if the relation is true (for ex. MAX_TOOLS = 100).
- The keywords EXIST or NOTEXIST, followed by the name of a machine group or by a string containing the name of a machine group or the name of a library, are specified after the IFDEF instruction. In this case the condition is satisfied if a group with the same name exists or doesn't exist in the Machine Configuration.
- After the IFDEF instruction LINKED or UNLINKED key word followed by the name of a device is specified. In this case the condition is verified, if the device is connected (LINKED) or not connected (UNLINKED) in virtual-physical. The device name can be expressed under this form: Group_Name.Subgroup_Name.Device_Name or Group_Name.Device_Name or Subgroup_Name_DeviceName or Device_Name. If the device does not exist in the configuration a compilation error appears.

It is possible to set more IFDEF instructions, remembering that each IFDEF instruction must correspond to an ENDEF instruction.

Example 1

```
; GPL code execution changes if the FRESA group is present in the machine
Const FresaGroup = "Fresa"
IFDEF Exist FresaGroup
 instruction
 instruction
ELSEDEF
 instruction
 instruction
ENDDEF
```

Example 2

```
; GPL code execution changes according to the module
IFDEF _ID_MODULE = 1     ; compile instruction for module 1
 instruction
 instruction
ELSEDEF        ; compile instruction for the other modules
 instruction
 instruction
ENDDEF


; compile code for the 2.4.10 version of Albatros
IFDEF _VER_MAJOR   = 2
 IFDEF _VER_MINOR = 4
      IFDEF _VER_REVISION = 10
            instruction
            instruction
      ENDDEF
 ENDDEF
ENDDEF


; compile code for the service pack 10 version of Albatros
IFDEF _VER_SP   = "Service Pack 10"
 instruction
ENDDEF


; compile code only if the system is configured for a Clipper
module
IFDEF _REMOTE_MODULE  = 1   ; 1 = Clipper, otherwise 0
 instruction
ENDDEF

; compile code for the 2.4 version service pack 10 Albatros
IFDEF _VER_FULL   = $0002040AH
 instruction
ENDDEF
```

Example 3

```
; the execution of the GPL code changes
; if the device is connected in virtual-physical
IFDEF LINKED  out1   ; if Out1 is connected, the code is executed
 istruction
 instruction
 instruction
ENDIF
```

MESSAGE

Syntax

MESSAGE **number [, IDposiz [, arg1, …, arg3]]**

Arguments

number	constant or variable
IDposiz	constant or variable. Numeric value used in synoptics.
arg1, …, arg3	constant or device or variable.

Description

It generates a message for the operator.
It generates a message for the operator. The parameter **number** can identify a module cycle error (i.e. a whole number) or group cycle error (in this case, DEFMSG applies. An argument, indicated by **IDposiz,** can also be passed if required. It indicates in which synoptic window the message should be displayed. It must correspond to the value specified in the synoptic creator for that specific

display cell. Albatros uses this identifier to handle messages in separate queues. A new queue is created for each IDposiz. If the IDposiz parameter is not specified, the message is set in the default queue with the value IDposiz=0. The range of the values that can be set is included between 0 (NOPLACE) and 1023. Albatros keeps a queue of the last 10 messages of the specified queue and of the last 100 messages of the default queue. When the messages queue is full, the latest message is overwritten. If the previous message of the queue is the same as the one that is going to be sent, the message is not sent (same task, same number, same argument).
The optional **arg1, ..., arg3** parameters are used to define parameter messages. The message's definition string will feature markers that will be replaced - when the message is generated - with the value or name of the device or variable passed as a parameter. Markers to be inserted in the string are as follows:

- $1, ... $2 replaced with the ***name*** of the device or variable ($1 stands for arg1 etc.)
- $(1), ..., $(3) replaced with the ***value*** of the device or variable.

Types of data valid for the arg1, ..., arg3 parameters are as follows:

- CHAR
- INTEGER
- FLOAT
- DOUBLE (though it is automatically converted into FLOAT)
- message number (or DEFMSG label)
- device
- global or local variable
- function parameter. It can be used as function parameter the label defined by the DEFMSG instruction.

Two error management modes are defined and established by manufacturer of the machine:
Messages managed like warning signals: all messages are sent. Albatros keeps a queue of the last 10 messages of the specified queue and the last 100 errors of the default queue. When the message queue is full it overwrites the oldest message. If the previous message is identical to the one to be sent, the message is not sent (same task, same number, same argument).
Messages managed like statuses: message is considered active or inactive. If active, any further sending of the same message (by MESSAGE instruction) is ignored..

Strings, matrices and vectors cannot be used as parameters (although individual vector or matrix elements are valid). For local variables, only the value can be decoded, not the name.
For the purpose of deleting a message with the DELMESSAGE instruction, the arg1, ...arg3 parameters are disregarded.
See also instructions DELMESSAGE and CLEARMESSAGES.

Example 1

```
DEFMSG    MSG_TOOL    "Change the tool"
DEFMSG    MSG_TOOL_P  "Tool number $(1) loaded"

; tag for synoptic views
CONST           TOOLCHANGE = 7

; message shown in the Errors Bar or in not tagged sinoptic views' cells
MESSAGE   MSG_TOOL

; message shown in the Errors Bar and in sinoptic views' cells
; tagged with code 7
MESSAGE   MSG_TOOL, TOOLCHANGE

; message with parameters
MESSAGE   MSG_TOOL_P, NOPLACE, MxTools[3].Cod
```

Example 2

```
    ; defined in a group
    DEFMSG      MSG_CARICO  "loading"
```

```
Function ShowMessage
    MsgTool  MSG_CARICO MxUtensili[3].Cod
fret

; defined in a library
DEFMSG      MSG_TOOL    "Tool: $(1) $2"


Function MsgTool public
PARAM parameter1 as integer
PARAM parameter2 as integer

       MESSAGE MSG_TOOL NOPLACE parameter1 parameter2
fret
```

SYSFAULT

Syntax

SYSFAULT

Arguments

no arguments

Description

It disables the SYSOK signal.
This signal is disabled to indicate that the machine is not secured (for ex. the GPL that manage emergencies are not in execution).
See also instruction SYSOK.

SYSOK

Syntax

SYSOK **[nameoutput1 [, ... nameoutput8]]**

Arguments

nameoutput1 [...nameoutput8] name of digital output device

Description

Indicates to the numerical control which are the outputs are connected to the safaty circuits of the machine (it can be an output connected to a safety relay, which controls the power supply of the machine). The outputs are activated when the numeric control has completed machine booting and has activated all emergency management tasks. At this stage the machine can be considered safe. Up to a total of 8 digital outputs can be defined. On each remote one output can be enabled. Only the outputs available on the CN2004 can be enabled and not those available on the remotes connected to it. The list of the outputs declared in the first use of SYSOK instruction cannot be changed during the possible next sysok calls, until the control has been initialized. If the instruction is executed without parameters, the signal of SYSOK is restored.
See also instruction SYSFAULT.

TYPEOF

Syntax

TYPEOF **name, result**

Arguments

name	name of device, constant, functionname, variable, vector, matrix or matrix row
result	integer variable. Type of the first argument

Description

It returns the **name** type argument to the **result** variable.

WATCHDOG

Syntax

WATCHDOG	**status**

Arguments

status	predefined constant. Acceptable values are: **ON, OFF**

Description

This instruction enables the use of the watchdog connected to the TMSWD-Hardware. It allows you to identify error situations occurring while executing the GPL code.
To enable the use of Watchdog, assign ON to the parameter **status.**
To upgrade the counter of the board, assign ON to the parameter **status**. If you do not upgrade, the watchdog starts and the TMSWD deactivates the emergency exit of the machine.
To finish the use of Watchdog, assign OFF to the parameter **status.**

This instruction can only be used with TMSbus+, TMSCan+ and TMSCombo+ boards with FPGA 2.0 or higher and mounted TMSWD hardware module.

Example

```
Function TestWatchDog autorun

   watchdog ON              ; enables the watchdog management

loop:

   watchdog ON              ; upgrades the counter of the board

   goto loop

fret
```

3.2.14 CANopen

TMSbus boards with CAN control

GETCNSTATE

Syntax

GETCNSTATE	**board, node, status**

Arguments

board	constant or variable integer. Board number
node	constant or variable integer. Number of the node
status	constant or variable integer.

Description

It returns the status of the NMT protocol for the **node** of the **board** CANOpen as shown. For further information about the meaning of these parameters, make reference directly the documentation concerning each single CANopen device.

GETSDOERROR

Syntax

GETSDOSERROR	**board, error**

Arguments

board	constant or variable integer. Board number (from 1 to 4)
Error	variable integer. Error code

Description

It returns the last **error** occurred, referred to the SDO communication for the **board** CANOpen as shown. For further information about the meaning of these parameters, make reference directly the documentation concerning each single CANopen device.

GETMNSTATE

Syntax

GETMNSTATE **board, status**

Arguments

board	constant or variable integer. Board number (from 1 to 4)
status	constant or variable integer.

Description

It returns the status of the NMT protocol for the master node of the CANOpen **board** as shown. For further information about the meaning of these parameters, make reference directly the documentation concerning each single CANopen device.

RECEIVEDPDO

Syntax

RECEIVEPDO **board, node, PDOnumber**

Arguments

board	constant or variable integer. Board number (1 to 4)
node	constant or variable integer. Number of the node
PDOnumber	constant or variable integer. Number of the PDO

Description

It reads the PDO content specified from **PDO number** for the mentioned node. This instruction is used to read asynchronous PDOs (i.e. those PDOs that in the canbus.def file are shown with "ASYNC" attribute).
The read data are copied in the respective device just as defined in the canbus.def file. (Make reference to the canbus.def file description).
This instruction can only be used with TMSCan and TMSCan+ boards.

Example

In a CANBUS.DEF file written in this way

```
....

 CN(1) RPDO=2ASYNC+2+1ASYNC TPDO=2+1ASYNC ;

 ...

VAR

 ...
     TPA.Byte1 AS %IB1.1.4 ;
 ...

END_VAR
```

To read the content in the third asynchronous PDO, you should enter in the file of the function this line of code:
RECEIVEPDO1,1,3
according to the description in the CANBUS.DEF file the content of the TPA.Byte1 device is copied

SENDPDO

Syntax

SENDPDO **board, node, PDOnumber**

Arguments

board	constant or variable integer. Number of the card
node	constant or variable integer. Number of the node
PDOnumber	constant or variable integer. PDO number

Description

It writes the specified PDO content from **PDO number** for the mentioned node. This instruction is used to write asynchronous PDOs (i.e. those PDOs that in the canbus.def file are shown with "ASYNC"

attribute). (Make reference to the canbus.def file description).
This instruction can only be used with TMSCan and TMSCan+ boards.

Example

In a CANBUS.DEF file written in this way

```
....

 CN(1) RPDO=2ASYNC+2+1ASYNC TPDO=2+1ASYNC ;

 ...

VAR

 ...
      TPA.Byte1 AS %IB1.1.2 ;
 ...

END_VAR
```

To send to the node the value contained in the TPA.Byte1 device, you should enter in the file of the function this line of code:

```
SENDPDO     1,1,2
```

SETNMTSTATE

Syntax

SETNMTSTATE	**board, node, status**

Arguments

board	constant or variable integer. Board number (from 1 to 4)
node	constant or variable integer. Number of the node
status	constant or variable integer.

Description

It sets the status of the NMT protocol for the **node** of the **board** CANOpen shown. If the value of the node is equal to 0 (zero) or higher than 126, setting is applied to all the existing and configured nodes on CANOpen channel. For further information about the meaning of these parameters, make reference directly the documentation concerning each single CANopen device.

Valye	**Protocol status**
1	Operational
128	Pre-Operational

Board CIF30

CANOPENDRIVER

Syntax

CANOPENDRIVER	**card, reserved, [error]**

Arguments

card	constant or variable. Number of the card
reserved	constant or variable. Reserved
[error]	variable. Error code

Description

It opens the communication channel between GPL and the CANopen Card. The second parameter is reserved for future use. The optional **error** parameter contains the codes of errors that could be generated during functioning; if it is not specified, in case of error, a system error occurs. See also CANCLOSEDRIVER.

CANCLOSEDRIVER

Syntax

CANCLOSEDRIVER	**card, [error]**

Arguments

card	constant or variable. Number of the card
[error]	variable. Error code

Description

It closes the communication channel between GPL and the CANopen Card. If the channel hadn't been opened it generates an error. The optional **error** parameter contains the codes of errors that could be generated during functioning; if it is not specified, in case of error, a system error occurs. See also CANOPENDRIVER

CANRESETBOARD

Syntax

CANRESETBOARD	**card, [error]**

Arguments

card	constant or variable. Number of the card
[error]	variable. Error code

Description

It executes the reset of the indicated CANopen Card. The optional **error** parameter, if specified, contains the codes of errors that could be generated during functioning; if it is not specified, in case of error, a system error is generated.

CANSETOBJECT

Syntax

CANSETOBJECT	**card, node, index, subindex, data, length, [error]**

Arguments

card	constant or variable. Number of the card
node	constant or variable. Number of the node
index	constant or variable. Index of objects folder
subindex	constant or variable. Subindex of objects folder
data	constant or variable. Data to be written
length	constant or variable. Length of data in bytes
[error]	variable. Error code

Description

It writes a CANopen object on the indicated **card**. The parameters **node**, **index** and **subindex** allow you to address the CANopen device and the location on which the CANopen object must be written. For further information about the meaning of these parameters, as well as the type and dimension of the data, consult directly the documentation concerning each single CANopen device.
The optional **error** parameter, if specified, contains the codes of the errors that could be generated during functioning; if it is not specified, in case of error a system error is generated. See also CANGETOBJECT.

CANGETOBJECT

Syntax

CANGETOBJECT	**card, node, index, subindex, data, length, [error]**

Arguments

card	constant or variable. Number of the card
node	constant or variable. Number of the node
index	constant or variable. Index of objects folder
subindex	constant or variable. Subindex of objects folder
data	variable. Data to be read
length	constant or variable. Length of data in bytes
[error]	variable. Error code

Description

It reads a CANopen object from the indicated **card**. The parameters **node**, **index** and **subindex** allow you to address the CANopen device and the location from which the CANopen object must be read. For further information about the meaning of these parameters, as well as the type and dimension of the data, consult directly the documentation concerning each single CANopen device. The optional **error** parameter, if specified, contains the codes of the errors that could be generated during functioning; if it is not specified, in case of error a system error is generated. See also CANSETOBJECT.

3.2.15 Mechatrolink II

MECCOMMAND

Syntax

MECCOMMAND **axis,command,parameters,reply,error**

Arguments

axis	name of digital axis device
command	integer constant.
parameters	integer array.
reply	integer array
error	integer variable. Error code

Description

It sends to indicated **axis** activation a **command** and waits for the reply. Necessary data for the execution of the command are inserted into **parameters** vector, while returned data from the execution of the instruction are stored into the **reply** vector. **Parameter** and **reply** vector must have the same size and the maximum number of elements must be 14. The consider value is the lowest byte of single integer. The **error** parameter contains the codes of eventual errors generated during the operation.

The error codes should be handled by Gpl as cycle errors.

The returned error codes are:

Error Codes	**Message**
-40	Command not allowed in the current functioning conditions
-41	Timeout error during the execution of a Mechatrolink II command
-44	Timeout error during the execution of a Mechatrolink II subcommand
-45	Link error of the drive

For the values that must be assigned to parameters **command**, **parameters**, **reply** and **error** see Yaskawa Mechatrolink II official documentation, where the values to be allocated to the command are described in the index 2 up to the index 15. The values to be set to the subcommands are described in the index 18 up to the index 32.

Commands can be distinguished in the following way:

- command. They have code includes between 0x00 and 0xFF. Because of safety reasons they are executed only if servo axis is enabled.
- subcommand. The commands used as subcommands must add to documented value the code 0x100. For example the command NOP has documented code 0x00, used as subcommand is 0x100.
- procedure. The commands used as procedures have command with value starting from 0x200. Currently those procedures are contemplated:
  - $201H habilitation procedure for offline parameters (to use with disenabled axis)

This instruction can only be used with Albmech, Dualmech and Dualmech Mono boards. For further information about the use of this instruction contact T.P.A. S.p.A

Note

This instruction acts on the actions of digital axes and it should be used in controlled context.

MECGETPARAM

Syntax

MECGETPARAM **axis,parameter,dimension,data,error**

Arguments

axis	name of digital axis device
parameter	constant or integer variable.
dimension	constant or integer variable.
data	integer variable.
error	integer variable. Error code

Description

It reads a parameter of the activation of indicated **axis** and it stores the parameter into **data** variable. The **error** parameter contains the codes of the possible errors generated during the operation. The error codes should be handled by Gpl as cycle errors.
The returned error codes are:

Error Codes	**Message**
-40	Command not allowed in the current functioning conditions
-41	Timeout error during the execution of a Mechatrolink II command
-44	Timeout error during the execution of a Mechatrolink II subcommand
-45	Link error of the drive

For the values that must be assigned to **parameter** and **dimension** variables see Yaskawa Mechatrolink II official documentation.
This instruction can only be used with Dualmech and Dualmech Mono boards. For further information about the use of this instruction contact T.P.A. S.p.A

MECGETSTATUS

Sintax

MECGETSTATUS **axis,state,inout,error**

Arguments

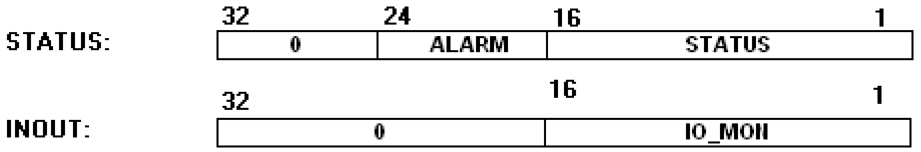
axis	name of digital axis device
state	constant or integer variable.
inout	constant or integer variable.
error	integer variable. Error code

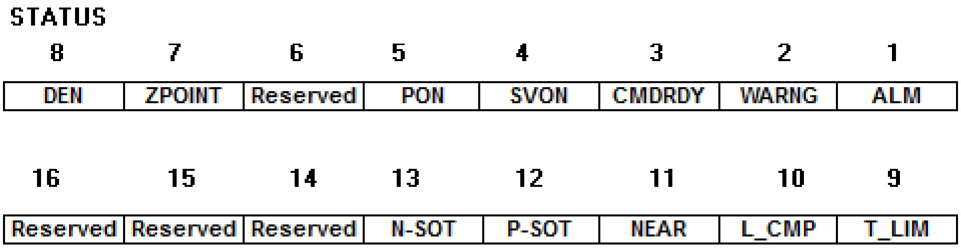
Description

It reads and stores into **state** variable the value of STATUS and ALARM and into **inout** variable the value of IO_MON relative to specified **axis**. For the values of STATUS, ALARM, IO_MON see Yaskawa Mechatrolink II official documentation.
The **error** parameter contains the codes of the possible errors generated during the operation. The error codes should be handled by Gpl as cycle errors.
The returned error codes are:

Error Code	**Message**
-40	Command not allowed in the current functioning conditions
-41	Timeout error during the execution of a Mechatrolink II command
-44	Timeout error during the execution of a Mechatrolink II subcommand
-45	Link error of the drive

A sequence of error categories is defined. The category that represents the value of the highest nibble of ALARM.
into one of following categories 0x30,0x70,0xD0,0xF0 must be sent a command of CLEAR (0x06).
Alarms that are included into one of following categories 0x00,0x10,0x40,0xB0 can’t be deleted with a command. It is necessary to solve the problem that creates the alarm, turn out the servodriver and switch it on again.
The structure of variables **state** and **inout** is a mask of bit structured as in the following representation:





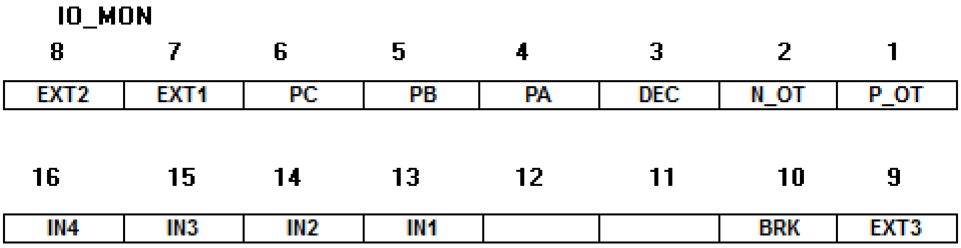
STATUS

8	7	6	5	4	3	2	1
DEN	ZPOINT	Reserved	PON	SVON	CMDRDY	WARNG	ALM

16	15	14	13	12	11	10	9
Reserved	Reserved	Reserved	N-SOT	P-SOT	NEAR	L_CMP	T_LIM

Meaning of STATUS bits

Bit	Command	Physical pins that can be connected in Virtual-Physical
1	ALM (Alarm)	Digital input
2	WARNG (Warning)	Digital input
3	CMDRDY (Command Ready)	
4	SVON (Servo ON)	Digital output
5	PON (Main Power ON)	Digital input
6	Reserved	
7	ZPOINT (Zero Point)	
8	PSET (Position Complete)	
9	DEN (Command Distribution Completed Flag)	
10	T_LIM (Torque Limit)	
11	L_CMP (Latch Completed)	
12	NEAR (Position Proximity)	
13	P-SOT (Forward-direction Software Limit)	
14	N-SOT (Reverse-direction Software Limit)	
15	Rerserved	
16	Reserved	



IO_MON

8	7	6	5	4	3	2	1
EXT2	EXT1	PC	PB	PA	DEC	N_OT	P_OT

16	15	14	13	12	11	10	9
IN4	IN3	IN2	IN1			BRK	EXT3

Meaning of IO_MON bits

Bit	Command	Physical pins that can be connected in Virtual-Physical
1	P_OT (Forward Over Travel)	
2	N_OT (Reverse Over Travel)	
3	DEC (Deceleration Limit Switch)	

4	PA (Phase A)	
5	PB (Phase B)	
6	PC (Phase C)	Digital input
7	EXT1 (First external latch input)	Digital input
8	EXT2 (Second external latch input)	Digital input
9	EXT3 (Third external latch input)	
10	BRK (Brake output)	
11		
12		
13	IN1 (General-purpose input 1)	
14	IN2 (General-purpose input 2)	
15	IN3 (General-purpose input 3)	
16	IN4 (General-purpose input 4)	

This instruction can only be used with Albmech, Dualmech and Dualmec Mono boards. For further information about the use of this instruction contact T.P.A. S.p.A

MECSETPARAM

Syntax

MECSETPARAM **axis,parameter,dimension,data,error**

Arguments

axis	name of digital axis device
parameter	constant or integer variable.
dimension	constant or integer variable.
data	integer variable.
error	integer variable. Error code

Description

It writes a **data** into the **parameter of indicated axis**.
For the values that must be assigned to **parameter** and **dimension** variables see Yaskawa Mechatrolink II official documentation. The **error** parameter contains the codes of the possible errors generated during the operation. The error codes should be handled by Gpl as cycle errors.
The returned error codes are:

Error Code	**Message**
-40	Command not allowed in the current functioning conditions
-41	Timeout error during the execution of a Mechatrolink II command
-44	Timeout error during the execution of a Mechatrolink II subcommand
-45	Link error of the drive

This instruction can only be used with Albmech, Dualmech and Dualmech Mono boards. For further information about the use of this instruction contact T.P.A. S.p.A

Note

This instruction acts on the actions of digital axes and it should be used in controlled context. To input data into into the non-volatile memory the instruction MECCOMMAND is to be used.

3.2.16 PowerlinkII and EtherCAT

Instructions to initialize the Powerlink nodes

Syntax

GETCNSTATE **board, node, state**

GETMNSTATE	**board, state**
SETNMTSTATE	**board, node, state**

Description

Make reference to the documentation in the chapter TMS boards with CAN control.

AXCONTROL

Syntax

AXCONTROL **axis, data**

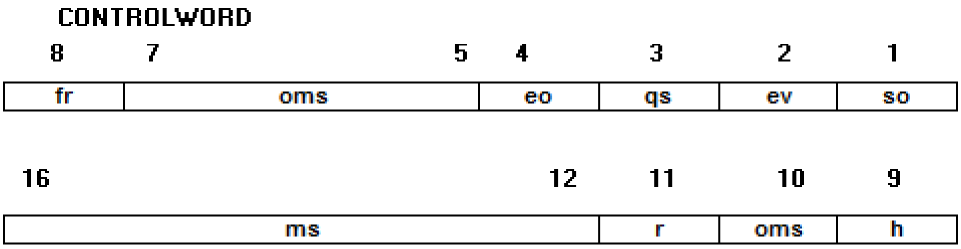
Arguments

axis	device name of axis type
data	variable or integer constant. it sets the ControlWord

Description

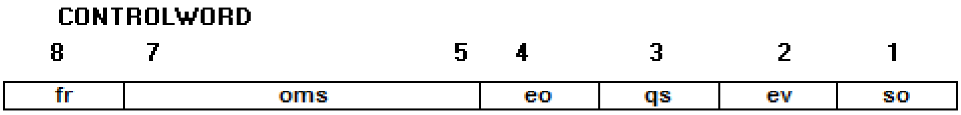
It sets the ControlWord **data**, in conformity with the functioning operativity, according to "CiA 402 CANopen device profile".

PowerLink and EtherCAT value definition table



Bit	Meaning	Name in virtual-physical
1	so=Switch ON	CW1
2	ev=Enable voltage	EV
3	qs=Quick stop	STOP
4	eo=Enable operation	SVON
5	oms=Operation mode specific	CW5
6	oms=Operation mode specific	CW6
7	oms=Operation mode specific	CW7
8	fr=Fault reset	RESALM
9	h=Halt	CW9
10	oms=Operation mode specific	CW10
11	r=Reserved	CW11
12	ms=Manufacturer specific	CW12
13	ms=Manufacturer specific	CW13
14	ms=Manufacturer specific	CW14
15	ms=Manufacturer specific	CW15
16	ms=Manufacturer specific	CW16

Table to define the values for S-CAN



CONTROLWORD

8	7 5	4	3	2	1
fr	oms	eo	qs	ev	so

Bit	Meaning	Name in virtual-physical
1	Ten_cmd=torque enable command 1:torque axis 0:free axis	SVON
2	Ien_cmd=movement enable command 1:enabled movements 0:axis stall	ENMOVE
3	Stp_cmd=stop command 1:active stop command 0:non-active command stop	STOP
4	Alm_rst= alarm status 1:alarm command reset	RESALM
5	Ltc_rst: reset bit 5 of StatusWord	CW5
6	oms=selected mode specific	CW6
7	oms=selected mode specific	CW7
8	oms=selected mode specific	CW8

ACTIVATEMODE

Syntax

ACTIVATEMODE **axis, data, err**

Arguments

axis device name of axis type
data constant or integer variable. Operating mode
err integer variable. Error code not returned by the servocontrol

Description

Sets the operating mode defined in the **data** variable according to "CiA 402 CANopen device profile". The operating mode of the starting axis corresponds to the **data** value = 9, that is "Synchronous speed configuration". The instruction returns **err=** 0 value, if the command succeeded, otherwise it returns an error code.
Given below, the table of the values to assign to data to choose the operating mode.

Value	Definition
+6	Homing mode
+9	Cyclic sync velocity mode

AXSTATUS

Syntax

AXSTATUS **axis, value**

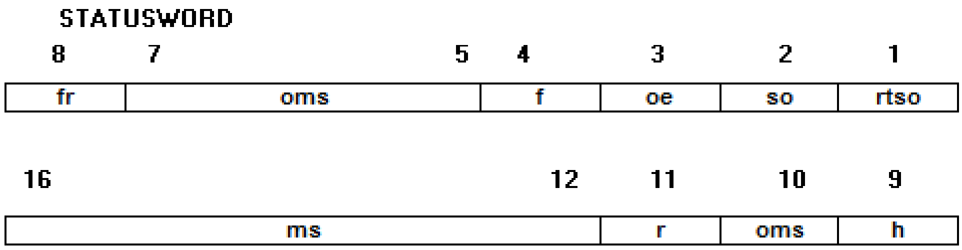
Arguments

axis device name of axis type
value integer variable

Description

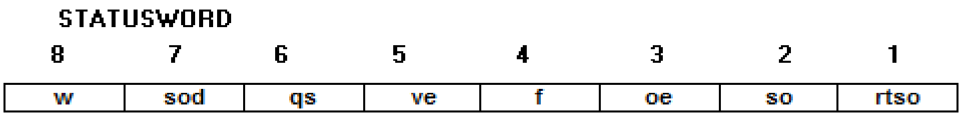
It return the value in the StatusWord in accordance with "CiA 402 CANopen device profile".

PowerLink II and EtherCAT value definition table



Bit	Meaning	Name in virtual-physical
1	rtso=Ready to switch on	RTSO
2	so=Switched on	SW2
3	oe=Operation enabled	OE
4	f=Fault	ALM
5	ve=Voltage enable	VE
6	qs=Quick stop	QS
7	sod=Switch on disabled	SOD
8	w=Warning	WARNG
9	ms=Manufacturer specific	SW9
10	rm=Remote	SW10
11	tr=Target reached or reserved	SW11
12	ila=Internal limit active	SW12
13	oms=Operation mode specific	SW13
14	oms=Operation mode specific	SW14
15	ms=Manufacturer specific	SW15
16	ms=Manufacturer specific	SW16

S-CAN value definition table



Bit	Meaning	Name in virtual-physical
1	Ten_st=torque enable status 1:torque axis 0:free axis	SW1
2	Ien_st=movements enable status 1:enabled movements	SW2

	0:axis stall	
3	Stp_st=stop status 1:running stop ramp 0:stop is not activated or ramp finished	SW3
4	Alm_st=alarm status 1:alarmed machine 0:no alarm detected	ALM
5	Ltc_st=Position latch status 1:position latch executed, register ready to read 0:no position latch detected	SW5
6	oms=operation mode specific	SW6
7	oms=operation mode specific	SW7
8	oms=operation mode specific	SW8

CNBYDEVICE

Syntax

CNBYDEVICE **device, board,cn**

Arguments

device device name
board integer variable. Board number returned
cn integer variable. CN-number returned

Description

Returns the EPL-coordinates of the device defined in the **device** parameter. This instruction can be used for instructions without direct connections to devices, as, for instance READDICTIONARY and WRITEDICTIONARY, that are directly connected to the EPL network.

Note

To further information concerning this instruction, see "CiA 402 CANopen device profile".

GETPDO

Syntax

GETPDO **board,node,nPDO,nObj,data,[error]**

Arguments

board constant or variable integer. Board number
node constant or variable integer. Position helt by the slave in the EtherCAT chain (from 1 on)
nPDO constant or variable integer. PDO identifier (ex, $1600h) or position of the same in the list of configured PDO configured in the ECATBIS.DEF file for the node under consideration (from 1 to 8)
nObj constant or variable integer. Object identifier (ex. $6040h) or position of the same within the list of object configured in the PDO (from 1 to 8)
data variable integer. It receives the value.
Error variable integer. Error code

Description

It returns in **[data]** the content of an object exchanged through the PDOs configured for the EtherCAT node. If the passed arguments are wrong and if the **error,** parameter has not been set, a system error is generated. If an **error** parameter has been set, this will contain the numeric code for the corresponding system error.

HOMING

Syntax

HOMING **axis, data, speed, speed1,offset,err**

Arguments

axis device name of axis type

data	constant or integer variable. Kind of homing
speed	constant or float or double variable. Search speed of the switch
speed1	constant or float or double variable. Search speed of zero
offset	constant or float or double variable. Zero - offset beside the homing position
err	integer variable. Error code returned by the servocontrol

Description

This instruction can be used in Powerlink II configuration only.
It runs the "zero search" according to the DS402 specifications. To know if the kind of **data** homing is supported by the servodrive, it is necessary to make reference to the producer's specifications. At the end of the homing operations, CN is placed into the previous operating mode.

Note

To further information concerning this instruction, see "CiA 402 CANopen device profile".

READDICTIONARY

Syntax

READDICTIONARY **board,cn,index,subindex,dimdata,data,err**

Arguments

board	constant or integer variable. Board number
cn	constant or integer variable. CN number
index	constant or integer variable. Object's index in the dictionary
subindex	constant or integer variable. Object's subindex in the dictionary
dimdata	integer variable. Dimension of the read data
data	char variable, integer, float,double,string. Variable receiving the data
err	integer variable. Error code returned by CN

Description

It reads the content of an objects' dictionary object, contained in CN. The instruction enables to read by means of the SDO protocol all the objects defined in accordance with "CiA 402 CANopen device profile" beside all the other objects made available by the manufacturer. To know the measning of the **index**, **subindex** and **dimdata** parameter, reference is made to "CiA 402 CANopen device profile" or to the specifications of the CN manufacturer. For the S-CAN devices the sub-index parameter must always be set to zero.

SETPDO

Syntax

SETPDO **board,node,nPDO,nObj,data,[error]**

Arguments

board	constant or variable integer. Board number)
node	constant or variable integer. Position helt by the slave in the EtherCAT chain (from 1 on)
nPDO	constant or variable integer. PDO identifier (ex, $1600h) or position of the same in the list of configured PDO configured in the ECATBIS.DEF file for the node under consideration (from 1 to 8)
nObj	constant or variable integer. Object identifier (ex. $6040h) or position of the same within the list of object configured in the PDO (from 1 to 8)
data	variable integer . Set value
Error	variable integer . Error code

Description

It sets the content **[data]** of an object exchanged through the PDOs configured for the EtherCAT node. If the passed arguments are wrong and if the **error,** parameter has not been set, a system error is generated. If an error parameter has been set, this will contain the numeric code for the corresponding system error.

WRITEDICTIONARY

Syntax

WRITEDICTIONARY	**board,cn,index,subindex,dimdata,data,err**

Arguments

board	constant or integer variable. Board number
cn	constant or integer variable. CN number
index	constant or integer variable. Object's index in the dictionary
subindex	constant or integer variable. Object's subindex in the dictionary
dimdata	constant or integer variable. Dimension of the data to write
data	char variable, integer,float,double,string. Variable containing the data
err	integer variabile. Error code returned by CN

Description

It writes the content of an objects' dictionary object, contained in CN. The instruction enables to read by means of the SDO protocol all the objects defined in accordance with "CiA 402 CANopen device profile" beside all the other objects made available by the manufacturer. To know the meaning of the **index**, **subindex** and **dimdata** parameter, reference is made to "CiA 402 CANopen device profile" or to the specifications of the CN manufacturer. For the S-CAN devices the sub-index parameter must always be set to zero.

3.2.17 SLM

SLMCOMMAND

Syntax

SLMCommand	**axis, command [,error]**

Arguments

axis	name of digital axis device
command	integer variable. Code of command to be executed. Possible values range between 0-255
error	integer variable. Error code

Description

It executes an SLM command. Any execution errors can be managed by GPL by means of an "Error" optional parameter. If the **error** variable has not been defined, in case of error it generates a system error.
This instruction can be used only with digital axis cards.
For further information about the use of this instruction contact T.P.A. S.p.A.

SLMEEPROMDISABLE

Syntax

SLMEEPROMDISABLE	**axis, [,error]**

Arguments

axis	name of digital axis device
error	integer variable. Error code

Description

It executes the write disabling command of an EEPROM memory location. It also returns any possible protocol errors that may be managed by GPL by means of the optional **error** parameter. If the **error** variable has not been specified, in case of error it generates a system error. This instruction can be used only with digital axis cards.
For further information about the use of this instruction contact T.P.A. S.p.A.

SLMEEPROMENABLE

Syntax

SLMEEPROMENABLE	**axis, [,error]**

Arguments

axis	name of digital axis device
error	integer variable. Error code

Description

It executes the write enabling command of an EEPROM memory location. It also returns any possible protocol errors that may be managed by GPL by means of the optional **error** parameter. If the **error** parameter has not been specified, in case of error it generates a system error. This instruction can only be used with digital axis cards.
For further information about the use of this instruction contact T.P.A. S.p.A.

SLMGETEEPROM

Syntax

SLMGetEEPROM **axis, address, data [,error]**

Arguments

axis	name of digital axis device
address	integer variable. Location to be read. Possible values range between 0-128
data	integer variable. Data returned from read
error	integer variable. Code error

Description

It executes the reading of an EEPROM memory location. Any execution errors can be managed by GPL by means of the optional "Error" parameter. If the **error** variable has not been defined, in case of error it generates a system error.
This instruction can be used only with digital axis cards.
For further information about the use of this instruction contact T.P.A. S.p.A.

SLMGETPARAM

Syntax

SLMGetParam **axis, address, data [,error]**

Arguments

axis	name of digital axis device
address	integer variable. Location to be read. Possible values range between 0-128
data	integer variable. Data returned from read
error	integer variable. Code error

Description

It executes the reading of an SLM parameter. Any execution errors can be managed by GPL by means of the optional "Error" parameter.
This instruction can be used only with digital axis cards.
For further information about the use of this instruction contact T.P.A. S.p.A.

SLMGETREGISTER

Syntax

SLMGetRegister **axis, register, data [,error]**

Arguments

axis	name of digital axis device
register	integer variable. Number of SLM register. Possible values range between 1-16
data	integer variable. Data returned from read
error	integer variable. Code error

Description

It executes the reading of the specified SLM register. Any execution errors can be managed by GPL by means of the optional "Error" parameter.
This instruction can be used only with digital axis cards.
For further information about the use of this instruction contact T.P.A. S.p.A.

SLMGETSTATUS

Syntax

SLMGetStatus	**axis, parameter, data [,error]**

Arguments

axis	name of digital axis device
parameter	integer variable. Read address (for ex. 8000h). Possible values range between 0-65535
data	integer variable. Data returned from read
error	integer variable. Code error

Description

It executes the write of the Multiax Read Address and returns the read parameter to the "Drive Status". It manages automatically a 1ms delay between two operations. Any execution errors can be managed by GPL through the optional "Error" parameter.
This instruction can be used only with digital axis cards.
For further information concerning the use of this instruction, contact T.P.A. S.p.A.

SLMSETEEPROM

Syntax

SLMSetEEPROM	**axis, address, data [,error]**

Arguments

axis	name of digital axis device
address	integer variable. Location to be written. Possible values range between 0-128
data	integer variable. Data to be written
error	integer variable. Code error

Description

It executes the write of an EEPROM memory location. Any execution errors can be managed by GPL through the optional "Error" parameter.
This instruction can be used only with digital axis cards.
For further information concerning the use of this instruction, contact T.P.A. S.p.A.

SLMSETPARAM

Syntax

SLMSetParam	**axis, address, data [,error]**

Arguments

axis	name of digital axis device
address	integer variable. Location to be written. Possible values range between 0-128
data	integer variable. Data to be written
error	integer variable. Code error

Description

It executes the writing of an SLM parameter. Any execution errors can be managed by GPL through the optional "Error" parameter.
This instruction can be used only with digital axis cards.
For further information concerning the use of this instruction, contact T.P.A. S.p.A.

SLMSETREGISTER

Syntax

SLMSetRegister	**axis, register, data [,error]**

Arguments

axis	name of digital axis device
register	integer variable. Number of SLM register. Possible values range between 1-16
data	integer variable. Data to be written

error	integer variable. Code error

Description

It executes the writing of the specified SLM. Any execution errors can be managed by GPL through the optional "Error" parameter.
This instruction can be used only with digital axis cards.
For further information concerning the use of this instruction, contact T.P.A. S.p.A.

3.2.18 Simulation

DISABLE

Syntax

DISABLE	**axis1,[...axis6]**

Arguments

axis1...[...axis6]	name of axis devices

Description

It disables the specified axes. This allows you to carry out simulations of the machine cyclic without physically moving the axes. A disabled axis can not read the information coming from the encoder but simulates a loop error proportionally to current speed. Disabling the axis, however, does not disable the speed reference, implying that power on the axes connector will not equal zero during simulated movements. For this reason it is necessary to disconnect the controls from the power supply or from the axis card during simulated movements, that is when axes are disabled. See also ENABLE.

Note

Step-by-step axes can be used in this instruction only if they are controlled by a TRS-AX remote.

DISABLEFORCEDINPUT

Syntax

DISABLEFORCEDINPUT

Arguments

no arguments

Description

It disables the possibility of using functions to force the inputs. If any inputs have been previously forced, executing this instruction resets the real state. See also ENABLEFORCEDINPUT, DISABLEFORCEDINPUT, SETFORCEDINPUT,RESETFORCEDINPUT, SETFORCEDBCD, SETFORCEDPORT, SETFORCEDANALOG.

ENABLE

Syntax

ENABLE	**axis1,[...axis6]**

Arguments

axis1...[...axis6]	name of axis devices

Description

It enables the specified axes. The axes are always enabled in the initialization phase. This instruction is only called if the axes were previously disabled by a DISABLE instruction.

Note

Step-by-step axes can be used in this instruction only if they are controlled by a TRS-AX remote.

ENABLEFORCEDINPUT

Syntax

ENABLEFORCEDINPUT

Arguments

no arguments

Description

It enables input forcing. Before using instructions to enable or disable forced input devices, it is necessary to execute this instruction. Otherwise the input forcing instructions have no effect.
See also DISABLEFORCEDINPUT, SETFORCEDINPUT,RESETFORCEDINPUT, SETFORCEDBCD, SETFORCEDPORT, SETFORCEDANALOG.

RESETFORCEDINPUT

Syntax

RESETFORCEDINPUT	**nameinput**

Arguments

nameinput	name of digital input

Description

It forces to OFF the input specified in **nameinput**.
To use this instruction it is necessary to have already enabled input forcing, with the ENABLEFORCEDINPUT instruction.
See also DISABLEFORCEDINPUT, SETFORCEDINPUT, SETFORCEDBCD, SETFORCEDPORT, SETFORCEDANALOG.

SETFORCEDANALOG

Syntax

SETFORCEDANALOG	**analoginput, value**

Arguments

analoginput	name of analog input device
variable	constant or integer or float or double variable

Description

It forces the **value** of the analog input specified in **analoginput**.
To use this instruction it is necessary to have first enabled input forcing, using the ENABLEFORCEDINPUT instruction.
See also, DISABLEFORCEDINPUT, SETFORCEDINPUT,RESETFORCEDINPUT, SETFORCEDBCD, SETFORCEDPORT.

SETFORCEDBCD

Syntax

SETFORCEDBCD	**namedigit1, [namedigit2,...], variable**

Arguments

namedigit1,[namedigit2...]	name of nibble device
variable	constant or integer or char variable

Description

It converts the **variable** decimal value into a sequence of digits. Each digit is then converted in binary system and the bit mask thus obtained is forced in the relative input nibble.
The highest weight digit is associated to the first nibble (**namedigit1**).
To use this instruction it is necessary to have first enabled input forcing, using the ENABLEFORCEDINPUT instruction.
See also DISABLEFORCEDINPUT, SETFORCEDINPUT,RESETFORCEDINPUT, SETFORCEDPORT, SETFORCEDANALOG.

SETFORCEDINPUT

Syntax

SETFORCEDINPUT	**nameinput**

Arguments

nameinput	name of digital input

Description

It forces to ON the input specified in **nameinput**.
To use this instruction it is necessary to have first enabled input forcing, using the ENABLEFORCEDINPUT instruction.
See also DISABLEFORCEDINPUT, RESETFORCEDINPUT, SETFORCEDBCD, SETFORCEDPORT, SETFORCEDANALOG.

SETFORCEDPORT

Syntax

SETFORCEDPORT	**portname, value**

Arguments

portname	name of input port device
variable	constant or integer or char variable

Description

It forces the value in the input port indicated by portname. The input port is interpreted as a bit mask.
If a bit equals 1 the relative input is forced to "ON".
To use this instruction it is necessary to have already enabled input forcing, with the ENABLEFORCEDINPUT instruction.
See also DISABLEFORCEDINPUT, SETFORCEDINPUT,RESETFORCEDINPUT, SETFORCEDBCD, SETFORCEDANALOG.

3.2.19 Blackbox

The purpose of the "BlackBox" functionality is to record in a database all the activities of a machine, that is a local or a remote module. The" activity of a machine" is the variation over time of a subgroup of all logic devices that can be used in GPL. This is the way to analyse afterwards the behaviour of the machine, linking the states of the stored devices. The database has a table containing a temporal information and the state of all devices in that time, one for each column. In the GPL language new instructions have been introduces to start, and and query for the recording activity and are described later.

Each file of blackbox is a SQLite database and it contains information concerning one only module. The file name includes the number of the module, the date and the time of the recording start.

Records are added in the database in a transactional manner. Each transaction contains at the most the records generated in 1 second. In the event of a power failure the coherence of the file is guaranteed and the last transaction can be lost. The maximum duration of the transaction can be modified by an entry in Tpa.ini (for further information, please contact T.P.A. S.p.A).

A limit of 12 hours to the duration of the recording has been inserted. This means that each database will always contain only the last 12 hours of recording. During the recording the most ancient records are removed from the database. The maximum duration of the history recorded in the database can be modified by an entry in Tpa.ini (for further information, please contact T.P.A. S.p.A).

This functionality is available for physical devices on GreenBus, EtherCAT, CAN, S-CAN and Mechatrolink II buses, connected through TMSbus, TMSbus+, TMScan, TMScan+, DualMech, DualMech Mono and AlbMech.

ENDBLACKBOX

Syntax

ENDBLACKBOX

Description

It ends the record on file functionality for all the activity of a local and remote module. See also STARTBLACKBOX and PAUSEBLACKBOX.

PAUSEBLACKBOX

Syntax

PAUSEBLACKBOX

Description

It pauses the file logging functionality of all the activity of a local or remote module. To resume the recording you need to carry out the instruction STARTBLACKBOX instruction without arguments. See also ENDBLACKBOX.

STARTBLACKBOX

Syntax

STARTBLACKBOX **[value][,error]**

Arguments

value	constant or variable integer. Recording period
error	variable integer. Error code

Description

It activates the file recording functionality of all the activity of a local or remote module. The activity of a module is the variation over time of the state of the logic devices excluding the flag switch, input nibble and output nibble devices.

Recording period (**value**) is expressed in milliseconds. It cannot be less than 10 and it must be a multiple of the realtime period. Otherwise, the system error no. 4399 (Parameter outside the range) would be generated.

If the instruction starts a record and the **value** is omitted, the considered default value is 20.

If the instruction resumes a previously interrupted record, no set **value** is considered.

If it was not possible to start the recording, **error** contains a value not equal to 0, otherwise it contains 0.

Error code	**Description**
0	No errors
1	There are some differences between the device configuration in the numeric control and the device configuration in Albatros
2	The number of the devices to record exceeds the maximum number provided for the system
3	No devices in the configuration
4	The communication software in the remote module does not support the blackbox functionality (remote modules only)
5	The numeric control prevents the recording from being started
6	Error in uploading the database management library
7	The number of columns for the table exceeds the maximum number of columns that can be managed by the database
8	Couldn't open the database on disc
9	Couldn't create in the database the recording table
10	Error in IP address for the communication with the remote module (remote modules only)
11	Couldn't create the communication socket to receive the data (remote modules only)
12	Couldn't associate a local address to the communication socket (remote modules only)
13	Couldn't connect to the remote socket (remote modules only)
14	Couldn't access to the memory region shared with the numeric control
15	The hardware configuration prevents from using the "BlackBox" functionalities
16	The functionality has been disabled in tpa.ini

See also PAUSEBLACKBOX and ENDBLACKBOX.

3.2.20 ISO

ISOG0

Syntax

ISOG0 **label, axis1 position1, axis2, position2, axis3, position3, axis4, positiona4,axis5, position5, [value]**

Arguments

label	constant or variable integer. Label identifying a displacement bloc. N in the ISO

standard.

axis1	device name of axis type. (X in the ISO standard)
position1	constant or variable Position of axis1 operational space
axis2	device name of axis type. (Y in the ISO standard)
position2	constant or variable Position of axis2 operational space
axis3	device name of axis type. (Z in the ISO standard)
position3	constant or variable Position of axis3 operational space
axis4	device name of axis type. (C in the ISO standard)
position4	constant or variable Position of axis4 joint space
axis5	device name of axis type. (B in the ISO standard)
position5	constant or variable Position of axis5 joint space
value	constant or variable double. It represents the feed rate percentage. F in the ISO standard

Description

It sets the rapid movement. The rapid movement sections are managed as synchronized. The points defined by the user are the extrema of the single space of displacement covered, so that all the axes are synchronized to each other. That means that the physical axes move individually, even though they start and arrive simultaneously, in the same way as in the instructions MULTIABS and MULTIINC. The tool point does not cover a line in the operational space and its trajectory is not checked. The parameter **label** is used in association with the instruction SETLABELINTERP to identify univocally the displacement bloc. The first three **positions** identify the position of the point in the operational space, while the following two positions define the value of the rotating axes in the joint spaces. The feed rate **value** defines the percentage of reduction as regards the most possible speed rate (In ISO: F0 highest speed, F100 FeedRate null, therefore the axes are still).

The instruction generates a system error (4105- Instruction not executable on axis AxisName), if used on step-to-step axes.

The instruction WAITCOLL cannot be used, because starting from the collision the interpolation link to the other axes that contribute to the movement and generate a profile other than that expected, would be get lost.

If used, the system error no. "4101 - Inconsistent axis AxisName management" is generated".

ISOG1

Syntax

ISOG1 **label, axis1, position1,axis2, position2, axis3, position3,axis4,position4,axis5, position5, [value]**

Arguments

	constant or variable integer. Label identifying a displacement bloc. (N in the ISO standard)
axis1	device name of axis type. (X in the ISO standard)
position 1	constant or variable Position of axis1 operational space
axis2	device name of axis type. (Y in the ISO standard)
position2	constant or variable Position of axis2 operational space
axis3	device name of axis type. (Z in the ISO standard)
position3	constant or variable Position of axis3 operational space
axis4	device name of axis type. (C in the ISO standard)
position4	constant or variable Position of axis4 operational space
axis5	device name of axis type. (B in the ISO standard)
position5	constant or variable Position of axis5 operational space
feed rate	constant or variable double. it represents the Feed value. (F in the ISO standard)

Description

It defines the point in the operational space that should reach the tool point at the end of the interpolation of the current bloc. The parameter **label** is used in association with the instruction SETLABELINTERP to identify univocally the displacement bloc. The first three **positions** identify the position of the tool point in the operational space, while the following two positions define the value of the rotating axes in the configuration space. The **value** Feed defines the feed rate of the tool point as measure unit (millimeters or grades) per minute (set in the presence of an instruction ISOG94) as inverse of the execution time (in the presence of the instruction ISOG93). The parameter **value** is compulsory for the first instruction ISOG1 of the interpolation movement.

The instruction generates a system error (4105- Instruction not executable on axis AxisName), if used on step-to-step axes.

The instruction WAITCOLL cannot be used, because starting from the collision, the interpolation link to the other axes that contribute to the movement and generate a profile other than that expected, would be get lost.

If used, the system error no. "4101 - Inconsistent axis AxisName management" is generated".

ISOG9

Syntax

ISOG9 **axis**

Arguments

axis name of device of type axis

Description

It enables the forced stop of the movement. If this instruction is active, the interpolation or the rapid movement are stopped before jumping to the next bloc. However, it is not a blocked instruction, like the instruction WAITSTILL . The control is informed about a forced stop and the capture process of the movement blocs proceeds up to the filling of the lookahead. The parameter **axis** finds the interpolation channel with 5 axes to be stop at the end of the bloc calculated before. In this case there is no difference if an instruction ISOG1 on an instruction ISOG0 is performed.

ISOG90

Syntax

ISOG90 **axis**

Arguments

axis name of device of type axis

Description

It sets the interpretation of the positions as absolute positions. The parameter **axis** finds the interpolation channel with 5 axes, that from this instruction on will interpret the axes positions as absolute positions (default condition). In this case there is no difference if an instruction ISOG1 on an instruction ISOG0 is performed.

ISOG91

Syntax

ISOG91 **axis**

Arguments

axis name of device of type axis

Description

It sets the interpretation of the positions as relative positions The parameter **axis** finds the interpolation channel with 5 axes, that from this instruction on will interpret the axes positions as relative positions. In this case there is no difference if an instruction ISOG1 on an instruction ISOG0 is performed.

ISOG93

Syntax

ISOG93 **axis**

Arguments

axis name of device of type axis

Description

It sets the speed interpretation as inverse of the execution time. The parameter **axis** finds the interpolation channel with 5 axes, that from this instruction on will interpret the value arisen from the F-parameters of the instruction. ISOG1 as inverse of the execution time expressed in minutes. Thanks to this, the control is able to determinate the speed rate to be kept by the tool point in the interpolation blocs.

ISOG94

Syntax

ISOG94 **axis**

Arguments

axis	name of device of type axis

Description

It sets the interpretation of the speeds as units of measure per minute. The parameter **axis** finds the interpolation channel with 5 axes, that from this instruction on will interprete the speed rates as measure units per minute (default condition).

ISOG216

Syntax

ISOG216 **RotariesMatrixName, ToolHolderMatrixName, ToolsMatrixName,EnablingMask, axis1, axis2,axis3,axis4,axis5**

Arguments

RotariesMatrixName	name of the matrix. It contains the data concerning the rotary axes.
ToolHolderMatrixName	name of the matrix. It contains the data concerning the toolholders
ToolHoldersMatrixName	name of the matrix It contains the data concerning the tools.
EnablingMask	variable or integer constant. C and B axes enabling mask
axis1	device name of axis type. (X in the ISO standard)
axis2	device name of axis type. (Y in the ISO standard)
axis3	device name of axis type. (Z in the ISO standard)
axis4	device name of axis type. (C in the ISO standard)
axis5	device name of axis type. (B in the ISO standard)

Description

It identifies the three matrices for the machine parametrisation and the five devices of axis type composing the same. Such instruction **should** be performed before every other ISO instruction. The parameter **EnablingMask** defines which rotation axes (C and/or B) should be enabled. To set the values, reference is made to the following table:

EnablingMask	**Description**
31	Desabling C and B axes
23	Enabling the only B axis
15	Enabling the only C axis
7	Desabling C and B axes

Note

The unit of measure, in which the values of the rotary axes are expressed in the configuration, must be degrees.

The link among the physical axes and the virtual ISO axes, set through this instruction, is brought to the end through the instruction ISOM2 or when the task, where the instruction is defined, has finished. Therefore, the axes can be used for classic movement.

ISOG217

Syntax

ISOG217

axis1,axis2,axis3,axis4,axis5,virtualAxis1,virtualAxis2,virtualAxis3,virtualAxis4,virtualAxis5.

Arguments

axis1	device name of axis type

axis2	device name of axis type
axis3	device name of axis type
axis4	device name of axis type
axis5	device name of axis type
virtualAxis1	device name of virtual axis type (X in standard ISO)
virtualAxis2	device name of virtual axis type. (Y in the ISO standard)
virtualAxis3	device name of virtual axis type (Z in standard ISO)
virtualAxis4	device name of virtual axis type (C in standard ISO)
virtualAxis5	device name of virtual axis type (B in standard ISO)

Description

It describes the physical axes and the virtual axes, which make up the machine. The virtual axes describe position and orientation of the tool and must be declared as virtual type in Albatros configuration. The first five specified axes must be physical and are controlled by the interpolator. The next five must be virtual axes; they are the axes that are used in the instructions ISOG0 and ISOG1. This instruction **must** be be performed before every other ISO instruction.
The formulas of direct and inverse kinematics to switch from a position in the space of the joints (physical axes) to the operational space (virtual axes) must be specified through the instruction KINEMATICEXPR for each of the ten axes, defined in the instruction ISOG217.
The instruction generates a system error (4105- Instruction not executable on axis AxisName) if used on step-to-step axes.

Note

The link between the physical axes and the virtual ISO axes set through this instruction, is loosed when the task, where the instruction is defined, is brought to an end or when the instruction ISOM2 is performed. Therefore, the axes can be used for classic movement. The instruction generates a system error (4105- Instruction not executable on axis AxisName), if used on step-to-step axes

ISOM2

Syntax

ISOM2 **axis**

Arguments

axis	name of device of type axis

Description

It frees the axes free from ISO movement, set through the instruction ISOG216 or the instruction ISOG217

ISOM6

Syntax

ISOM6 **axis, RotaryMatricesRowIndex, ToolHolderMatrixRowIndex, ToolMatrixRowIndex**

Arguments

axis	name of the axis device
RotaryMatricesRowIndex	constant or variable integer. Row index of the rotary axes matrix
ToolHolderRowMatrixIndex	constant or variable integer. Row index of the matrix of the toolholder
ToolHolderRowMatrix	constant or variable integer. Row index of the matrix of the toolholders

Description

It sets the use of a group of parameters describing the machine's kinematics. The **indexes** refer to three matrices whose name is determined by the user. They are declared in the file of the global variables of Albatros. The axis **parameter** identifies the corresponding interpolation channel. How the three matrices in the file of the global variables should be declared, is described in the tables, as follows:

Matrix field	Matrix of rotary axes
X - Offset	Offset along X between the pivot point and the control point of the head
Y-Offset	Offset along Y between the pivot point and the control point of the head

Z-Offset	Offset along Z between the pivot point and the control point of the head
Out-of-alignment of X	Deviation in X between rotation and slewing axes (when the position of C-axis = 0)
Out-of-alignment of Y	Deviation in Y between rotation and slewing axes (when the position of C-axis = 0)
Out-of-alignment of Z	Nose-pivot point distance
δ - angle δ	Angle around Z for the correct placement of the head with respect of zero point machine.
γ - angle γ	Angle between rotation and slewing plane.

Matrix fields	Toolholder Matrix
PU X-Offset	Offset in X between the toolholder's coupling point to the motor and the tool's c oupling point to the toolholder (when the position of C-axis = 0 and vertical motor)
PU Y-Offset	Offset in Y between the toolholder's coupling point to the motor and the tool's coupling point to the toolholder (when the position of C-axis = 0 and vertical motor)
PU Z-Offset	Offset in Z between the toolholder's coupling point to the engine and the tool's coupling point to the toolholder (when the position of C-axis = 0 and vertical motor)
Angle α	Phase displacement angle between motor and toolholder axis (with respect to Z)
Angle β	Phase displacement angle between motor and toolholder axis (with respect to Y)

Matrix fields	Matrix of the toolholders
Length of the tool	Length of the tool

ISOSETPARAM

Syntax

ISOSETPARAM **ParameterIndexNumber, value**

Arguments

ParameterIndexNumber constant or variable integer. It is the number identifying a parameter

constant value or variable float. It is the value to set.

Description

It sets some parameters ruling the fluidity of the ISO interpolated movement. The meaning of each **ParameterIndexNumber,** the values within which the variable should be **included** and the values defaults are explained in the table, as follows:

ParameterIndexNumber	RANGE	Default	Meaning
0	0.0-100.0	50.0	Linear axes slowdown percentage in case of angular point (0= no slowdown, 100= maximum slowdown allowed by the interpolator)
1	0.0-100.0	50.0	Rotating axes slowdown percentage in case of angular point. (0= no slowdown, 100= maximum slowdown allowed by the interpolator)
2	0.5-1.0	0.9	Factor of speed reduction on curviliear abscissa in case of angular point. (1=no reduction, 100=maximum slowdown allowed)
3	0.0-100.0	60.0	Slowdown percentage in case of close

			discontinuities. (0=no slowdown, 100=maximum slowdown allowed by the interpolator)
4	0.0-100.0	10.0	Smooth percentage of the trajectory
5	0.2-1.0	0.2	Minimum dimension of the space to cover with only linear axes. The value is expressed in millimeters.
6	0.1-1.0	0.1	Minimum dimension of the space to cover with only linear axes. The value is expressed in millimeters.
7	0.0-100.0	100.0	Percentage of the applied minimum smooth value (0 = minimum value of invalid smooth, 100 = maximum percentage of the minimum smooth value)

KINEMATICEXPR

Syntax

KINEMATICEXPR **axis = expression**

Arguments

axis name of device of physical or virtual axis type
expression group of operators

Description

It allows you to define single expressions of direct and inverse kinematics. Before performing this instruction, the instruction ISOG217 describing the physical axes and the virtual axes, that make up the machine, must be called. For each axis defined in ISOG217 the instruction KINEMATICEXPR. must be called. The kinematics expression of an axis in the space of the joints (inverse kinematics) can be a function of

- variables
- constants
- coordinates of the axes in the operational space.

The kinematics expression of an axis in the operational space (direct kinematics) can be a function of

- variables
- constants
- coordinates of the axes in the space of the joints.

The expression **syntax** is the same as in the instruction EXPR, the only difference being that local variables cannot be used. Furthermore, axes of the same type as the axis, declared in **axis** and not declared in the instruction ISOG217 , cannot be used. E.g, if the kinematics of a virtual axis, already declared in the instruction ISOG217 is being defined, in the expression only the five physical axes, that are declared in the ISOG217 , can be used.

Example

```
ut as double       ; tool number
offsety as double ; offset Y nose fulcrum
offsetz as double ; offset Z nose fulcrum

Function ISO5Ax


     setval 100,ut
     setval 120.0,offsety
     setval 60.0,offsetz
     ; EXPLICIT KINEMATICS
     ISOG217 Rx Ry Rz Rc Rb X Y Z C B

     ; DEFINITION OF THE KINEMATICS EXPRESSIONS
     ; EXPLICIT INVERSE KINEMATICS Rx physical AXIS
     KinematicExpr Rx = X - 135 + ut * sin ( B ) * cos ( C )

     ; EXPLICIT INVERSE KINEMATICS RY physical AXIS
```

```
KinematicExpr Ry = Y + offsety + ut * sin ( B ) * sin ( C )

; EXPLICIT INVERSE KINEMATICS Rz physical AXIS
KinematicExpr Rz = Z + offsetz + ut * cos (B)

; EXPLICIT INVERSE KINEMATICS Rc physical AXIS
KinematicExpr Rc = C

; EXPLICIT INVERSE KINEMATICS Rb physical AXIS
KinematicExpr Rb = B
```

3.2.21 Instructions which can not be used with interrupt

The following instructions cannot be used in the functions called by ONFLAG, ONINPUT and ONERRSYS.
Their usage is not allowed in realtime tasks too.
instructions.

Instructions which, in turn, call a function on interrupt:

- ONFLAG
- ONINPUT
- ONERRSYS

Instructions which involve a wait:

- WAITINPUT
- WAITFLAG
- WAITACC
- WAITCOLL
- WAITDEC
- WAITREG
- WAITTARGET
- WAITWIN
- WAITSTILL
- WAITTASK
- WAITRECEIVE
- WAITPERSISTINPUT
- MULTIWAITFLAG
- MULTIWAITINPUT

Communication instructions:

- SEND
- RECEIVE
- CLEARRECEIVE
- COMOPEN
- COMCLOSE
- COMREAD
- COMREADSTRING
- COMWRITE
- COMWRITESTRING
- COMGETERROR
- COMCLEARRXBUFFER
- COMGETRXCOUNT

Following instructions involving axis movement:

- MOVINC
- MOVABS
- LINEARINC
- LINEARABS
- CIRCLE
- CIRCINC
- CIRCABS
- HELICINC
- HELICABS
- COORDIN
- MULTIABS
- MULTINC
- SETRIFLOC
- SETTOLERANCE

- RESRIFLOC
- SETPFLY
- SETPZERO
- SETINDEXINTERP
- STARTINTERP
- FASTREAD
- ENABLE
- DISABLE
- ENDMOV

ISO instructions:

- ISOG0
- ISOG1
- ISOG9
- ISOG90
- ISOG91
- ISOG93
- ISOG94
- ISOG216
- ISOG217
- ISOM2
- ISOM6
- ISOSETPARAM
- KINEMATICEXPR

Following instructions involving Powerlink II management:

- HOMING
- READDICTIONARY
- WRITEDICTIONARY

Digital axis card configuration instructions:

- SLMGETPARAM
- SLMSETPARAM
- SLMCOMMAND
- SLMGETSTATUS
- SLMGETEEPROM
- SLMSETEEPROM
- SLMGETREGISTER
- SLMSETREGISTER
- SLMEEPROMENABLE
- SLMEEPROMDISABLE

Instructions involving multitasking:

- SENDMAIL
- WAITMAIL
- ENDMAIL
- SENDIPC
- WAITIPC
- TESTMAIL
- TESTIPC

Instructions which imply a long processing time:

- SORT
- FIND
- FINDB
- MOVEMAT
- CANOPENDRIVER
- CANSERETBOARD

3.2.22 Instructions which are no longer available

CLEARSTOPDISABLE	it clears the field stop disabling counter
STOPDISABLE	it disables the field stop
STOPENABLE	it enables the field stop
IFSTOPDISABLED	test on disabled field stop
SPINDLE	it sets the speed of a winding block

SETPARINV	it sets the wireguide inversion parameters for winding
WINDING	it stops an axis
BRAKEENABLE	it enables break management
BRAKEDISABLE	it disables break management
SETPREARN	it sets a prestop position for negative direction movement
SETPREARP	it sets a prestop position for positive direction movement
LET	it calculates arithmetical expressions
SENDRECEIVE	it sends data outside with a confirmation request
SEED	it sets the seed for a sequence of random numbers.

3.3 Examples

3.3.1 Homing on Interrupt

```
;------------------------------------------------------------------------
;  Example of on the fly homing routine
;
;  The function executes the following operations:
;
;     1) It sets the axis by disabling software limits
;        and setting position on zero.
;     2) It checks that the sensor is not already on ON state.
;        If it is on ON, it moves the axis and waitsfor it to return
;        to OFF state. If this does not happen in 30 seconds
;        it generates an error message.
;     3) It sets the sensor search speed
;     4) It launches axis movement and enables "on the fly" homing
;        for the specified axis. When the interrupt is relesed,
;        the axis position is set on zero and movement to a disengage
;        position is started automatically.
;     5) It waits for the axis to reach the disengage position.
;     6) It resets axis limits
;
;
;   © T.P.A. S.p.A.
;------------------------------------------------------------------------
Function Fast_Homing

     ResLimPos    axis          ; Axis start-up
     ResLimNeg    axis
     SetQuote     axis,0

     IfInput      FastInput,OFF,Goto Continue    ; Test occupied sensor
     SetVel       axis,5                         ; Set disengage speed
     MovAbs       axis,30                        ; Move axis
     WaitInput    FastInput,OFF,30,Call Error    ; Test micro disengage
                                                 ; Error after TimeOut=30

     EndMov       axis                           ; Stop axis
     WaitStill    axis                           ; Wait for axis stop

Continue:
     SetVel       axis,10       ; Homing sensor search speed
     MovAbs       axis,-1000    ; Sensor search negative movement
     SetPFly      axis,ON,10,0  ; Interrupt attach
                                ; and set disengage position and speed
```

```
         WaitStill   axis         ; Wait for axis Stop

         SetLimPos   axis         ; Reset axis limits
         SetLimNeg   axis

         Fret

     ; subprocedure to send error messages
     Error:
         Error       ERR_SETP     ;Error signalled: impossible to proceed
         Ret
```

3.3.2 Axis movement server

```
     ;--------------------------------------------------------------
     ; Example of axis movement server:
     ;
     ;  The server moves the machine's axes
     ;  on behalf of other tasks.
     ;
     ;  The client tasks send their commands in the form of
     ;  messages (mails) to a postbox.
     ;
     ;  The server takes the commands from the box and executes them.
     ;
     ;  The requests are queued in the post box, so that
     ;  if a request arrives while the server is already
     ;  engaged, it is not lost, and will be dealt with as soon as possible.
     ;
     ;  The server is the only task to move axes. This avoids
     ;  conflicts.
     ;
     ;  The server is implemented by the Master_axes function.
     ;
     ;  An example of client is implemented by the Check_flag function.
     ;  This function checks  the state of a flag
     ;  periodically and  when it finds it on ON it sends the server
     ;  the axis homing execution command.
     ;  The flag will presumably be set on ON manually
     ;  by the operator, using for example the synoptic view.
     ;
     ;--------------------------------------------------------------

     ;-----------------------------------
     ; -- MACHINE GLOBAL CONSTANTS --
     ;-----------------------------------
     Const MBOX = 101    ; identifies the command post box

     Const SETP = 10     ; axis homing
     Const CHG  = 11     ; change tool
     Const FORO = 12     ; execute perforation


     ;---------------------
     ; --- AXES GROUP---
     ;---------------------

     ; definition of error messages
     Defmsg ERR_CMD "Axis group command unknown"

     ; --- Server ---
     Function Master_axes autorun

        local cmd as integer            ; command
```

```
   local position_X as double       ; position X perforation
   local position_Y as double       ; position Y perforation

loop:
   waitmail MBOX,cmd,position_X,position_Y    ; wait command

   ; When the command arrives we identify it
   ; and execute the required action
   Select cmd

   case SETP
             fcall homing_axes     ; Axis Homing
   case CHG
             fcall Change_tool     ; Execute tool change
   case FORO
             fcall Perforation position_X,position_Y  ; perforation in
                                                      ; specified position
   case else
             call error
   endselect

   endmail MBOX                    ; command execution notification
   goto loop                       ; wait for new command


   fret

; subprocedure for error message sending
error:
   error ERR_CMD
   ret


;-------------------------
; --- GENERIC GROUP ---
;-------------------------

; --- Client ---
Function Check_flag

loop:

   ifflag Setp_axes,OFF, goto loop           ; test flag state

   ; OK the flag is on ON, send command
   sendmail MBOX,WAITTACK,SETP,0.0,0.0

   resetflag Setp_axes                       ; reset flag

   goto loop                                 ; back to wait

   fret

   ; NOTICE THAT:
   ; - after the "SETP"command, the two parameters "position_X"
   ;   and "position_Y" must be specified even if it does not
   ;   make sense for the Homing operation.
   ;   Because the server can not know beforehand which command
   ;   it will receive,we must specify two values
   ;   of the type expected by the server,
   ;   in this case, two DOUBLE. The values to be set are "0.0" and "0.0".
   ; - the "WAITACK" parameter makes the client wait
   ;   for the server to conclude the command.
   ;   The client can continue its own execution only when the Server
   ;   has executed an ENDMAIL or has started processing a new
   ;   command (WAITMAIL).
```

3.3.3 Main Cycle with error management

```
;------------------------------------------------------------
; Hypothetical main function
; start machine and execute test cycle
;------------------------------------------------------------
Function Main
  OnErrSys        GestErrSys              ; enable error management

  StartTask       Emergencies             ; start
  StartTask       Processor
  Enableaxes

loop:
  IfFlag          Flag,OFF, ResetEmergencies
  .........
  Goto            loop
Fret


;-----------------------------------
; error management function
;-----------------------------------
Function GestErrSys
  Param nerror as integer
  Param task as function
  Param typedevice as device

  EndTask         task                                ; End Processor task
  If              nerror, >, 5, goto noerraxis        ; The first 5 errors
                                                      ; concern
                                                      ; axes
  ResetFlag       Flag
  Disableaxes

noerraxis:
Fret
```

3.3.4 Operations on strings

```
;---------------------------------------------------------------
; Example of string manipulation
;---------------------------------------------------------------
Function example
  Local       string1 as string
  Local       string2 as string
  Local       string3 as string
  Local       length as integer
  Local       position as integer

  SetString   "String of",string1         ; string1 now contains
                                          ; "String of"
  SetString   " test",string2
  AddString   string1,string2,string3    ; stringa3 contains
                                         ; "Test string"

  Search      string3,'t',position       ; position equals 2
  Search      string3,'Z',position       ; position equals -1

  Left        string3,7,string1          ; string1
                                         ; contains "String"

  Right       string3,2,string2          ; string2
                                         ; contains "va"

  Mid         string3,9,2,string3        ; string3
                                         ; contains "of"

  ControlChar 65,string1                 ; string1
                                         ; contains "A"

  Len         string3,length             ; length equals 2

  Str         length,string3             ; string3
                                         ; contains "2"

  Val         position,string1           ; string1
                                         ; contains "-1"

  AddString   "The result is",string1,string2

  ; string2 contains "The result is -1"

Fret
```

3.3.5 Sequential / Parallel Execution

```
;------------------------------------------------
; Example of a routine testing the Homing
; of a 3 axes machine to avoid any
; mechanical interference.
;
; The Homing of the single axes is implemented
; by functions whose text has been omitted.
; See example "Homing Routine".
;
; The Homing of the z axis is carried
; out first(as theoretically it can not be
; done with the others),
; when this is concluded, the X and Y axis Homing
; is executed simultaneously.
;------------------------------------------------

; message for the operator (translated in set language)
DefMsg      MSG_SETP    ITA   "Homing assi in corso ..."
                        ENG   "Homing in progress ..."

Function Homing

 Message     MSG_SETP               ; inform operator

 Fcall       HomingAxisZ            ; Homing of Z axis

 ; OK Z axis Homing is concluded
```

```
    StartTask   HomingAxisX            ; launch homing X and Y
    StartTask   HomingAxisY

    WaitTask    HomingAxisX            ; wait for task end
    WaitTask    HomingAxisY

    DelMessage  MSG_SETP               ; delete message
                                       ; for the operator

  Fret
```

3.3.6 Homing Routine

```
;-----------------------------------------------------------------
;  Example of axis setpoint routine
;
;  The function executes the following operations:
;    1) it disables the software axis limits
;    2) it sets the switch search speed
;    3) it moves the axis to an incremental position that
;       guarantees reaching the switch
;    4) it waits for the axis to release the switch
;    5) it stops the axis and waits for movement to end
;    6) it sets the speed (low) of the disengage switch
;    7) it makes the axis move backwards the sufficient space
;       to disengage the switch
;    8) it waits for switch disengage
;    9) it sets the new position for the axis
;   10) it resets default speed and software limits
;
;   © T.P.A. S.p.A.
;-----------------------------------------------------------------

Function Homing

   ResLimPos    axis         ; disable software limits
   ResLimNeg    axis

   SetVel       axis,10      ; set speed

   MovInc       axis,10000   ; move the axis

   WaitInput    Switch,ON    ; wait for switch

   EndMov       axis         ; stop axis
   waitStill    axis         ; wait for axis stop

   SetVel       axis, 0.1    ; set disengage speed

   MovInc       axis,-100    ; move axis

   WaitInput    Switch,OFF   ; wait for switch disengage

   SetQuote     axis,0       ; assign new position

   SetVel       axis         ; reset speed
   SetLimpos    axis         ; reset software limits
   SetLimneg    axis

Fret
```

3.3.7 Synchronized movement

```
;-------------------------------------------------------------------
;  Example of synchronized movement
;
;  A profile is generated using the instruction SYNCRO
;  profile is then executed using the instruction COORDIN.
;
;   © T.P.A. S.p.A.
;-------------------------------------------------------------------

const CH1 = 1          ; synchronized movement channel
const CAD = 4          ; frequency of generation/execution of the positions
(4 ms)


pMat[5000] as double:Qx double:Qy integer:index
pVar as integer

Function Sincro
   local ini as integer

   ; profile generation
   SyncroOpen               CH1, CAD, pMat, pVar, ON

   SyncroSetVel             CH1, X, 20, 20
   SyncroSetVel             CH1, Y, 20, 20

   SyncroMove               CH1, X, 100, Y, 100
   SyncroMove               CH1, X, 110, Y, 120
   SyncroMove               CH1, X, 140, Y, 130

   SyncroSetVel             CH1, X, 10, 10    ; change speed axis X

   SyncroMove               CH1, X, 150, Y, 160
   SyncroMove               CH1, X, 200, Y, 180

   SyncroStartMove          CH1
   SyncroClose              CH1

   ; profile execution
   setval                   1,ini
   Coordin                  pMat, CAD, UP, ini, pVar, $11b, X, 1, Y, 2

   WaitStill                X, Y

Fret
```

3.3.8 Iso movements

```
------------------------------------------------------------------
;  Example of ISO movement
;
;  A profile is generated using the instruction ISOG0 and ISOG1
;
;  © T.P.A. S.p.A.
;-------------------------------------------------------------------*

;  Declaration of ISO matrices
Matrix of rotary axes
MxRot[5] as double:off_X double:off_Y double:off_Z double:dis_X
double:dis_Y double:dis_Z double:delta double:gamma
; Toolholder matrix
```

```
MxPorta[1] as double:off_X double:off_Y double:off_Z double:alpha
double:beta
;  Tools matrix
MxTools[10] as double:ut double

Function  ISOInterpolation

   ; setting of standard values of machine parametrisation
   setval 90.0 MxRot[5].gamma
   setval 260.3 MxTools[10].ut
   setval MxTools[10].ut ut

   ; setting of parameters of algorithm
   IsosetParam 0 50
   IsosetParam 1 50
   IsosetParam 2 0.9
   IsosetParam 3 60
   IsosetParam 4 30

   ; machine settings: declares the three matrices used for
   ; the machine parametrisation
   ; and the physical axes used in the ISO movements.
   isoG216 MxRot MxTool MxHolder 31 X Y Z C B ; IMPLICIT KINEMATICS

   ; setting of group of parameters describing the machine's kinematics.
   isoM6 X 5 1 10 ; IMPLICIT KINEMATICS

   ; setting of the starting value
   setquote x 500
   setquote y 300
   setquote z 0
   setquote c 0
   setquote b 0
   setvel x
   setvel y
   setvel z
   setvel c
   setvel b
   setveli x y z c b

   ; profile execution
   isoG0 1001,X 998.0,Y 600.0,Z 0.0,C 90.0,B 45.0,50.0
   isoG1 1001,X 998.0,Y 600.0,Z 0.0,C 90.0,B 45.0,10000.0
   isoG1 1003,X 996.0,Y 600.0,Z 0.0,C 90.0,B 45.0,10000.0
   isoG1 1002,X 600.0,Y 600.0,Z 0.0,C 90.0,B 45.0,10000.0
   isoG1 1004,X 599.131759111665,Y 599.924038765061,Z 0,C 100,B
   45.0,10000.0
   isoG1 1006,X 598.289899283372,Y 599.69846310393,Z 0,C 110,B 45.0,10000.0
   isoG1 1005,X 597.5,Y 599.330127018922,Z 0,C 120,B 45.0,10000.0
   isoG1 1003,X 596.786061951567,Y 598.830222215595,Z 0,C 130,B
   45.0,10000.0
   isoG1 1002,X 596.169777784405,Y 598.213938048433,Z 0,C 140,B
   45.0,10000.0
   isoG1 1012,X 595.669872981078,Y 597.5,Z 0,C 150,B 45.0,10000.0
   isoG1 1011,X 595.301536896071,Y 596.710100716628,Z 0,C 160,B
   45.0,10000.0
   isoG1 1031,X 595.075961234939,Y 595.868240888335,Z 0,C 170,B
   45.0,10000.0
   isoG1 1102,X 595.0,Y 0.0,Z 0.0,C 180.0,B 45.0,10000.0
   waitstill X Y Z C B
fret
```

Tecnologie e Prodotti per l'Automazione s.p.a

Via Carducci 221
I - 20099 Sesto S.Giovanni (MI)
Tel. +39 02.36527550
Fax. +39 02.2481008
www.tpaspa.it