Albatros

3.2.3

Numeric control

Tecnologie e Prodotti per l'Automazione

Table of Contents

1 Introduction

1.1 How to use this manual

This user guide describes the functions of Albatros numeric control.
Thanks to the structure of the manual, getting to know the system and learning how to use it will be an easy task for the operator.

The main subjects of each section of the guide are:

- Albatros windows and tools.
- description of the typical architecture of an Albatros system.
- how to display the devices and operate on them with manual and diagnostic functionality, using the Synoptic window.
- how to display and modify Technological Parameters, Geometrical Parameters and Tool Parameters.
- how to display the devices and operate on them with manual and diagnostic functionality.

To avoid overcharging this guide, for further information concerning the use of the *mouse*, *menus* and *toolbar* and all the current operative functions of Windows, we refer the reader to Windows Operative System manuals.

1.2 Work windows

There are various types of work windows, depending on the kind of operations required, and more than one can be kept open at the same time.

The types of windows are the following:

Window	**Description**
Main	Albatros main window. It allows to call functions and contains all the other windows whose content depends on the specific application they represent.
Synoptic	it contains a graphic representation of the machine, or of parts of the machine, and allows to operate on them.
Technological Parameters	it enables to display and modify the technological parameters.
Tool Parameters	it enables to display and modify the tool parameters.
Diagnostic	it enables to display the status of the devices and, if possible, to operate on them.
System Errors	window containing the list of the most recent system errors. It is also possible to display cycle errors and messages.
System Configuration	it enables to display and modify the logical and physical devices of the machine.

2 System composition

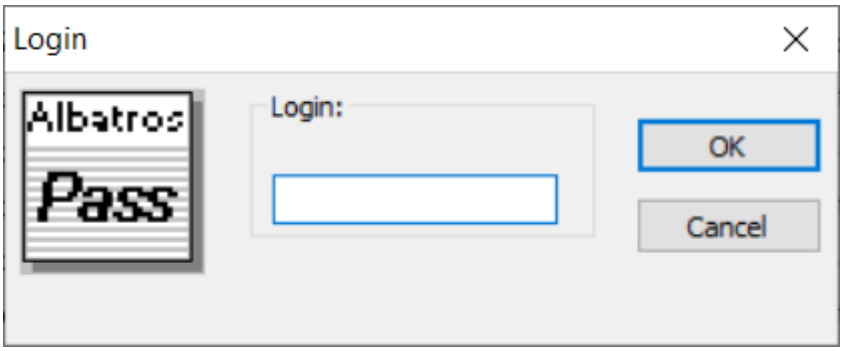
2.1 Access rights to the system

Albatros has four access levels to the system:

- User: is the level with most access restrictions. It does not allow to modify any of the device settings. It is the level used for machining and normal machine operations. When the system is booted, this access level is automatically enabled.
- Service: is the level used for ordinary maintenance of the machine. The operator should be able to modify some configuration parameters, without altering the structure of the machine.
- Manufacturer: is the level used to configure plants and machines. At this level almost any kind of modification is possible. It is used by developers.
- Tpa: is the highest access level of the system. Its function is to protect access to particularly delicate settings, whose modification requires a detailed knowledge of Albatros. This level is very rarely used and the access password must be requested directly at TPA.

To access the system at a higher level than User, or to return to User level after introducing changes at a higher level, the corresponding password must be introduced.

To recall the login window, press **Ctrl + *** (asterisk). Alternatively, click on the icon on the right of the Windows **Task bar** with the right mouse button to view a menu showing the **Change pass level** command. The window you are opening looks like this:



Login window

Now enter the password and press the **[OK]** key to confirm. The letters composing the word will be visualized as "*" characters, so that none can read the password typed in.
By typing in the password, you have logged into the corresponding access level. To have a confirmation of the level accessed, select the heading **About Albatros** from the **Help** menu.

If the password entered is not correct, the error message "Warning! Wrong Password!" will appear.

2.2 Multilingual support

Albatros supports the display of text in multiple languages.

Change language
The language may be changed at any access level of the system. To select a different language, you need to use the key combination **Ctrl + /** or click on the icon from the Windows "**task bar**".
In the window that will open, select the language required and click on **[OK]**.
The language change will not take place immediately, but at the following restart of Albatros.

2.3 Typical architecture of the system

Because many aspects of graphical representation and the structure of basic data of the Machine depend greatly on the kind of Machine, this Manual provides by way of example a description of the composition of a typical system, as well as some general information.

The detailed information, the diagrams and the graphics of the real system obviously depend on the specific application, and are consequently prepared by the Manufacturer of the Machine Tool.

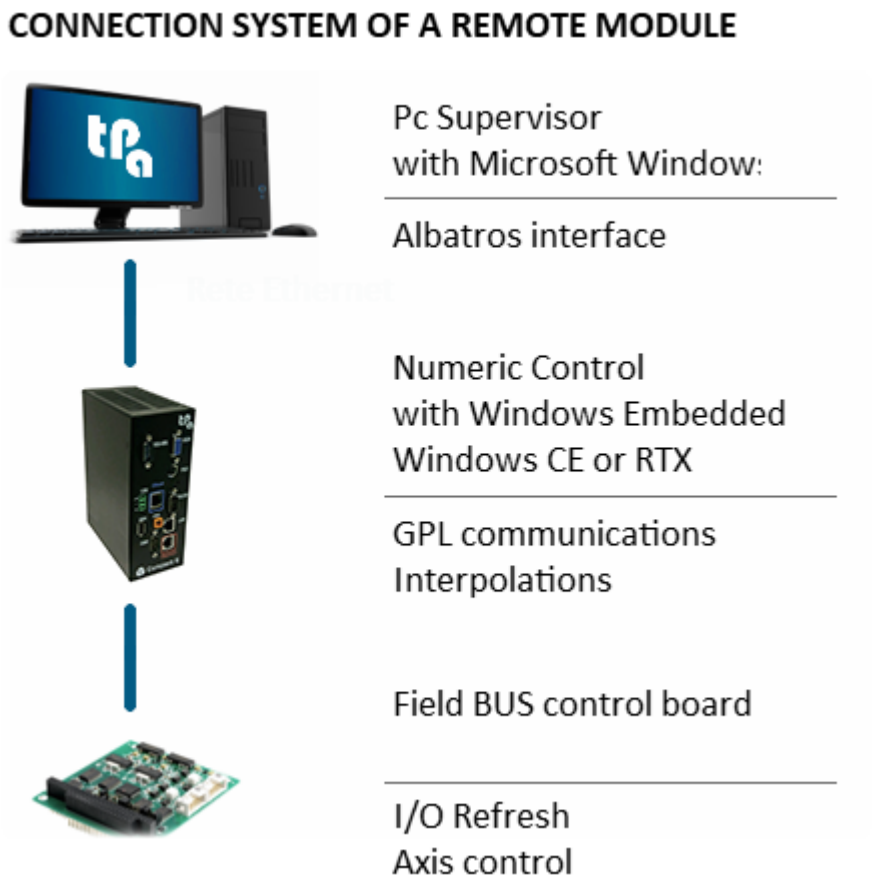
The Albatros numerical control system is composed of a supervisor PC, showing the Operator-Machine interface, and a number of Modules (range between 1 and 16) for the piloting and control of all operative resources of the Tool Machine or Plant.

So, you can have two kinds of plants:

Monomodule consisting of one module connected directly to the PC bus.

Multimodule consisting of a minimum of 1 and a maximum of 16 modules, usually used for applications on Plants or Lines with several machines; the PC unit in this case is physically separated from the Modules, which can be located in different points of the Line or Plant.

In both architectures, the modules are composed of one or more axis boards for the direct control of the Machine Axes and the logic management of the Input/Output system.
In the monomodule version, the axis boards are installed directly in the Supervisor PC, while in the multimodule version they are installed in an industrial PC (with or without screen and keyboard) connected to the Supervisor PC via Ethernet networl. The following figure shows the diagram of the connection between the Supervisor PC and the remote module (Clipper). The main activities of the single components are also described.



Intelligent remote devices pilot I/O devices and axes (TRS-AX remote) directly on the machine. These devices read the Digital Input (ON/ OFF) or Analog Input channels, refresh the Digital or Analog Output channels and are connected to the Modules by means of a GreenBUS (serial bus RS485 - 1 Mbaud), CAN bus, and EtherCAT. The profile machining of Albatros is protected by a USB hardware key, configured by TPA.

2.4 Organization and logic configuration

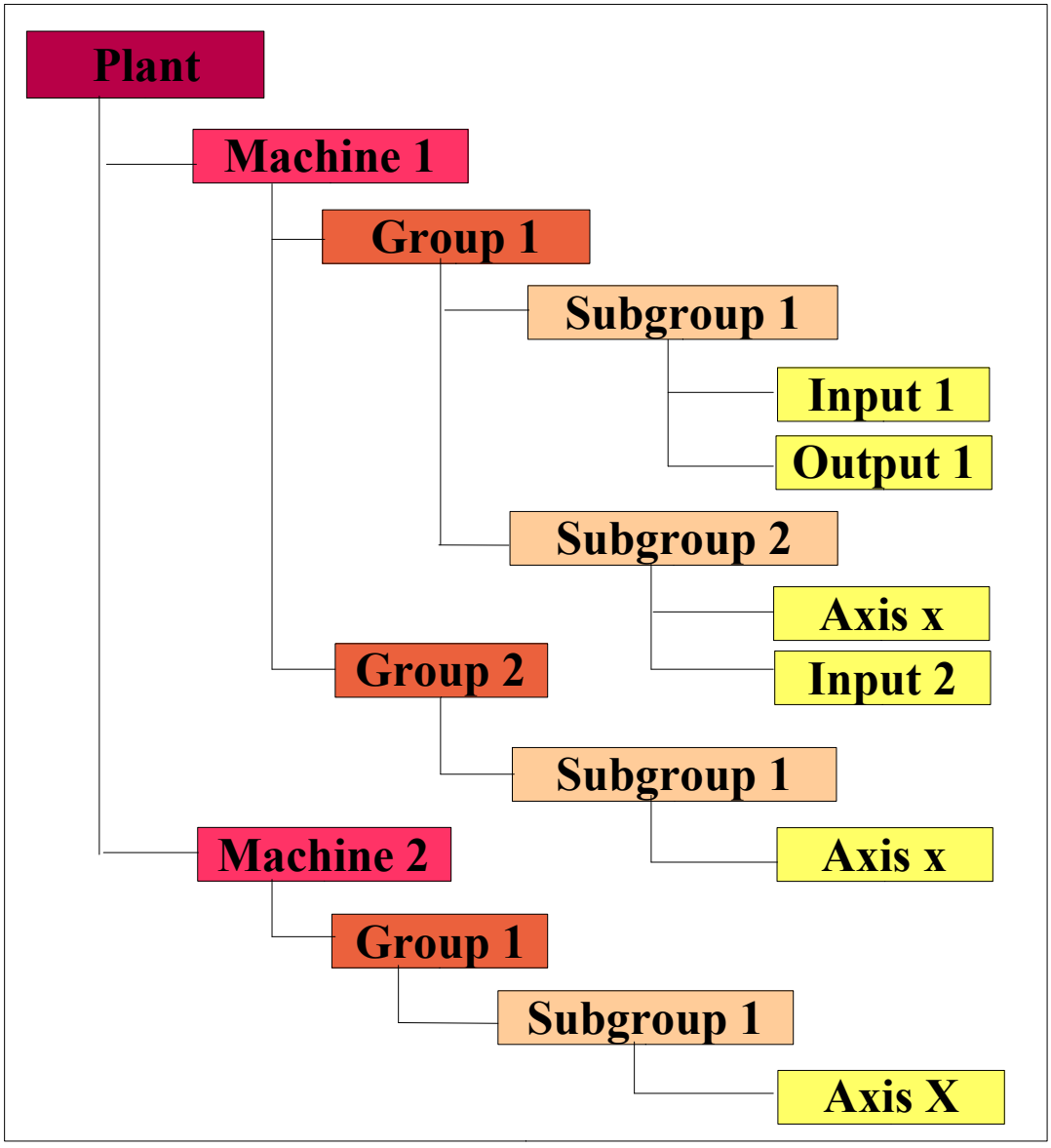
In the Albatros system, the descriptive structure of the plant or single tool machine is organised in a technological file with a hierarchical structure.

This approach allows, if necessary, to maintain the modular structure of the machine as far as the configuration data and access modalities are concerned, by classifying it in terms of dynamic association of various modules, aggregates and devices that may be enabled or disabled according to the required setting.

Following this logical structure, in the most general and complex case, the higher hierarchical level will be composed by:

1. **Plant** simply a set of machines. It represents the operational parts managed by the Numerical Control. The plant is always present, even in the case of a single machine and it is not necessary to mention it explicitly.
2. **Machine** from a "logic" point of view it is defined as a set of devices (axes, timer etc.) and control cycles, corresponding to a GPL language code that applies the control algorithms of the machine itself. Generally the machine is provided with a large number of devices which are organised into groups.
3. **Groups** are "containers" which allow to organise the components of the machine following a logical criteria. For example we could define an "axes" group containing all the axes of the machine, the limit switches, the cyclic performing the axis homing etc.
4. **Subgroups** indicate a further subdivision of a group. For example, the "axes" group could be divided into "digital axes" and "stepper axes".
5. **Devices** are the lowest level of the hierarchy. They are a logic representation of the electrical and mechanical components of the machine and are independent of the hardware below.

The following figure schematises the structure of a hypothetical plant composed of two machines:



Example of hierarchical structure of a plant.

NOTE: The Groups do not necessarily have to be divided into Subgroups, they can be directly made up of Devices.

In the case of a plant with more than one machine, to access given functions, such as Diagnostic, System configuration and Technological Parameters, it is necessary to select the machine whose data you need to view.

2.5 Devices

The devices can be grouped into two categories: physical devices and logical devices. In the system, all the devices are identified by a name describing their function.

Physical devices

By physical devices, we intend all those parts which act on the electrical or pneumatic parts of the machine or verify their status. These are:

Symbol	Device	Function
	*Digital input*	it verifies the status, "on" or "off", of a device. For example, the safety switch of a door.
	*Digital output*	it enables or disables a device, setting it on "on" or "off". It is used, for example, to pilot solenoid valves.
	*Analog input*	it measures the voltage of input power in the corresponding terminal. For example the power generated by a tachometer dynamo.
	*Analog output*	it defines the output voltage of the corresponding terminal. It can be used, for example, to pilot an inverter.
	*Input port*	it consists of 8 digital input channels.
	*Output port*	it consists of 8 digital output channels.
	*Axis*	it controls the movement of electrical axes. It is possible to control various kinds of axes: analogically controlled, digitally controlled, stepping motors, counting axes (only encoder reading).

Logical devices

Logical devices are parts which act exclusively within the operating programs and do not have a physical counterpart:

Symbol	Device	Function
	*Timer*	time counting device. The measurement unit is the second. Resolution: 4 ms. It can only indicate positive numbers, displaying a maximum time span of 8.589.934 seconds (with real-time at 250 Hz). The amount is recorded in the non-volatile memory of the axis board.
	*Counter*	operation counting device. It may display any number between -2.147.483.648 and +2.147.483.647. The amount is recorded in the non-volatile memory of the axis board.
	*Flag bit*	off/on indicator.
	*Flag switch*	special flags that can be connected to certain buttons on the tool bar, as the Start flag, for example.
	*Flag Port*	it is composed of 8 flag bit channels.
	*Variable*	GPL code *integer* type global variable.
	*Variable*	GPL code *char* type global variable.
	*Variable*	GPL code *float* type global variable.
	*Variable*	GPL code *double* type global variable.
	*Variable*	GPL code *string* type global variable.
	*Variable*	GPL code *array* type global variable.
	*Variable*	GPL code *matrix* type global variable.

3 Synoptic Panel

3.1 Using the Synoptic Panel

During the machine operation, it is possible to open the *Synoptic Panel* window to verify the status of the most important devices.

Synoptic panels display the same information contained in the diagnostic window. However, while in the latter the information is displayed in a tree structure (which includes all the devices present on the machine), synoptics illustrate the information graphically (displaying, for example, an image of the machine and setting the position of the axes next to the axes themselves). Synoptics also allow to select the most significant information, grouping the remaining information in secondary screen pages, to be recalled by the user when necessary.

3.2 How to operate on the Synoptic Panel

The Operator can select the various pages composing the synoptic, for diagnostic purposes, by *double clicking* with the mouse on one of the areas of the machine. The different areas are delimited by a dotted rectangle, and are also known as "hot spots".

To select a "hot spot", a device or an axis, simply move the mouse pointer on the image of the required object. The name of the selected device appears at the same time in the Status Bar.

The appearance of the mouse pointer changes according to the selected object, to indicate what kind of operation is allowed on that specific object. These are:

magnifying glass	if it is a "hot spot"
hand	if it is an output device
text cursor	if it is a set-value box

3.3 How to act on Devices

To act on a device, point the mouse on the required device, and complete the action as described below (actions vary according to the type of device).

Representation mode	**Action**	**Device**
Device icon	point and *click*	Digital output Flag switch Flag bit
Set-Value box	point, *click* and set value	Analog output Output port Flag port Axis position Timer Counter

3.4 Manual Axis Movement

To access the manual axis movement function, it is necessary to have the required access rights. Access rights are assigned by the manufacturer of the machine.

To interact with an axis, simply *double click* with the mouse on the positions display field of the required axis. The window for the axis movement will open. In the case of Virtual, Stepping motor and Count axes, the window contains less data. For example, in the case of a Count axis, only the Real Position and Speed values are displayed.

The window is composed of two areas, whose contents are described below.

Visualization area

- Three cells displaying the axis *Real Position [mm]*, its *Speed [m/min]* and the *Loop Error* or tracking error.

- Two select buttons which indicate the axis *Status* (*Free* = open loop, for example, because of a system error, *Normal* = closed loop, corresponding to normal position control status). It is also possible to set the status by using these buttons.
- During movement, the signalling of the axis *Status* (example: Acceleration).
- Two buttons to select negative or positive direction axis movement.
- The button**,** to Stop axis movement at any moment, during movement in Absolute or Step mode.

Movement area

- Two cells to set a *Negative Position* and a *Positive Position*, which will be used in *Absolute* mode.
- One cell to set the *Speed* set on the axis during manual movements.
- Three select buttons to choose what kind of movement to apply: *Jog*, *Absolute* position or *Step*.
- One cell to set the *Step* value to be used in *Step* mode.

To move an axis, set the parameters described above as required. Select the movement mode and press the button (to move the axis in positive direction) or the button (to move the axis in negative direction).
In *Jog* mode the axis will keep on moving as long as the or minus button is kept pressed.
In *Step* mode, the axis will move as far as indicated in the "Step" cell each time the or button is pressed.
In *Absolute* mode, the axis reaches directly the position indicated in the Positive position or Negative position cell.

It is also possible to use the keyboard "+" (or Ctrl+P), "-" (or Ctrl+M) and "Space bar", instead of the , and buttons.

4 Technological and Tool parameters

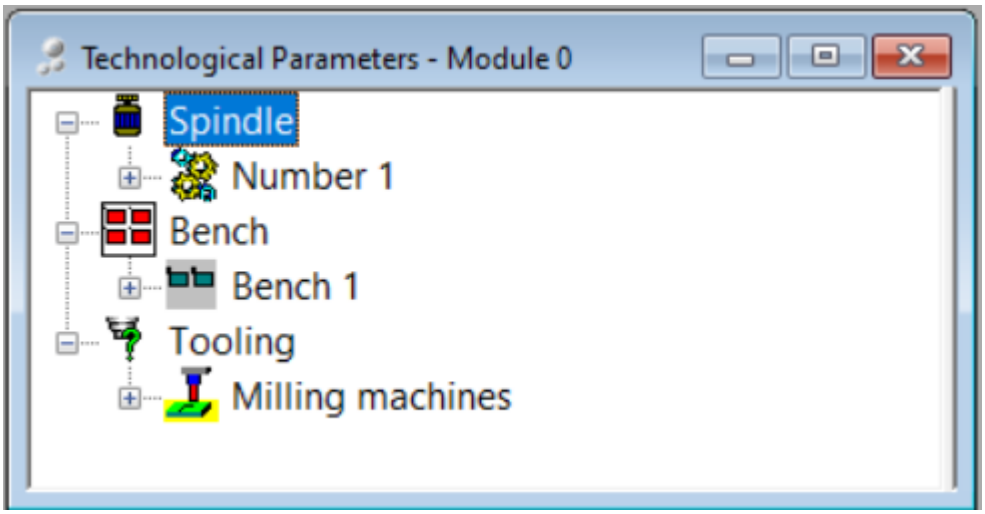
4.1 Technological Parameter Window

The Technological Parameter file allows to record the geometrical and technical information of a machine. The numeric control needs this information to handle machine functioning correctly.
To open: menu ***File->Open Technological Parameters***.

Technological Parameters are usually organized in Groups/Subgroups (normally the groups and subgroups of the Technological Parameters are independent of the groups and subgroups in which the machine devices are divided). The display modes are defined by the Machine Manufacturer and depend on the specific application.

The values listed in the file are usually set by the manufacturer in the Machine calibration phase and cannot be modified by the User, if not exceptionally. Therefore, some data may be protected by Password to avoid accidental modifications which could affect correct machine functioning.

The Technological Parameter window displays in a tree structure all the Groups and Subgroups of parameters that compose the file, as shown in the following figure.



Structure of the Technological Parameter file.

The window contains Groups, displayed in a tree structure, with their relative Subgroups of parameters. The tree structure can be expanded or collapsed using the ⊞ and ⊟ buttons found at each node. The +, -, and Right/Left arrow keys can also be used to open and close parts of the tree.

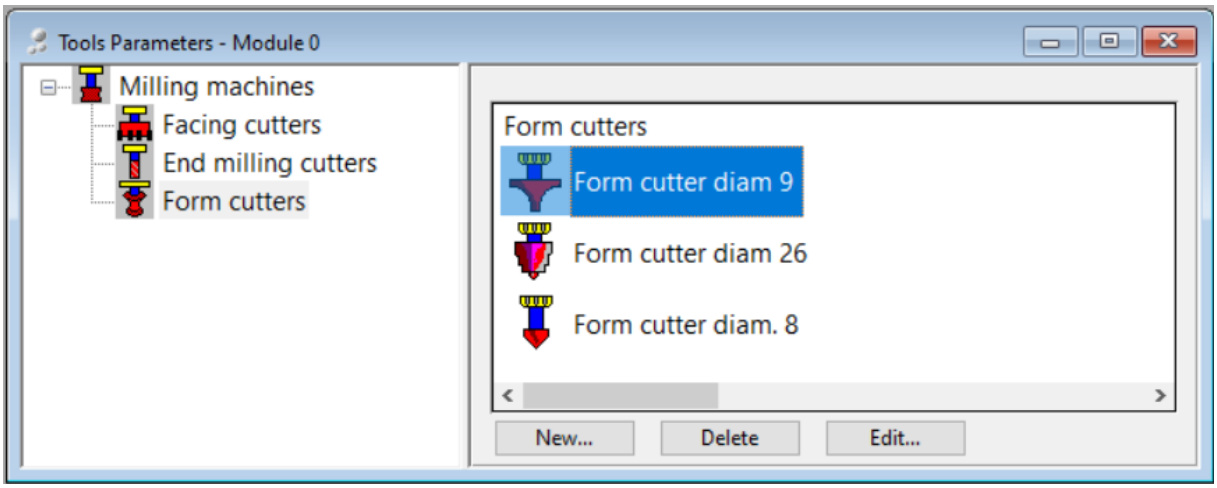
How to operate on Technological Parameters
Once the required Group/Subgroup tree is opened, it is possible to access the page containing the data.
The data can be listed in a table, or in text or selection cells, depending on the type of data and how the Manufacturer set the data.
If any data is modified, it is necessary to press **[OK]** to make the changes permanent.

Tooling
Tooling is an unusual kind of machine data. Typically, any information concerning the set of tools the machine is equipped with (tooling) is saved in the Technological Parameter file. However, any information concerning the tools themselves is saved in the Tools Parameter file. For this reason, to define the tooling of a machine, it is necessary to combine the information contained in the two files. If the system provides for this situation, it will be possible to recall information contained in the Tools Parameter file from the Technological Parameters file. Usually, the connection is implemented by means of a button with a similar icon to the one below.

Select the icon and double click on it with the mouse left button and a window containing the list of tools defined in the Tools Parameter file will open, allowing to select the required tool. When this has been done, the icon button changes, displaying the icon that represents the specified tool.
It is also possible to display the tool data by double clicking on the icon with the right hand button of the mouse.

4.2 Tool Parameter Window

The window of Tools Parameters can be opened in Menu ***File->Open Tool Parameters***.
Tool parameters, determined by the Manufacturer on the basis of the operations performed by the machine, are usually organised as shown in the figure below:



Example of a Tool Parameter Window.

The Tools Parameter window is divided in two areas:

- the *left area* contains the Groups, with the corresponding Subgroups of tools, displayed in a tree structure. The tree structure can be expanded or collapsed using the ⊞ and ⊟ buttons placed at each node. For example, we could have a Milling Cutters Group composed by Subgroups of cutters with different characteristics, such as profile milling cutters, traverse milling cutters etc. Each one of these subgroups is associated to one or more tools whose characteristics are assigned in a dialog window defined by the manufacturer. The tools contained in each subgroup are displayed in the right-hand side of the screen.
- the *right area* takes the name of the selected Subgroup and contains the list of the tools belonging to the Subgroup. The tools defined in this area do not necessarily exist on the machine. The association between the tool and the position on the machine (tooling) is normally done in the technological parameter file.

How to operate on Tool Parameters
Tools are added, modified and deleted from the file by means of buttons located in the lower section of the window:

[New...]	enables adding a new tool to the Subgroup. It opens the "New Tool" dialog window, in which the following data can be inserted: - *Description*: a message that identifies the tool. The description can be chosen from the ones already in the list, if it has not already been assigned to another tool, or a new description can be written. - *Image*: an icon that identifies the tool. It can be chosen from the ones already in the list, or it can be called from a folder using the **[Image]** button. The tool is inserted in the list following the alphabetical order of the descriptions.
[Delete]	allows removing a tool from the Subgroup, although it is subject to confirmation; the description of the tool is not deleted and remains available for another tool.
[Edit...]	allows replacing the *description* or *image* of the selected tool, through the same dialog window described in the **[New...]** command.

5 Diagnostics

5.1 The Diagnostics window

The Diagnostic window can be opened during machine execution to allow the operator to keep machine functioning under control, by monitoring the logic status of the I/O digital signals, analog I/O data, counters and timers data and axes movement.
Depending on the access rights conceded by the manufacturer, it may also be possible to modify the status of the devices.

If allowed by the access level, it is possible in real-time:

- to display the status (ON/OFF) of all the digital Input and Output signals.
- to able and disable the digital Output signals.
- to display the voltage (ranging between +/-10V) of the Analog inputs.
- to assign a voltage (ranging between +/-10V) to all the Analog outputs.
- to move an axis in Manual by selecting the speed, the Pitch or the final absolute Position, and display real position, speed and loop error.
- to display and modify the global variables.

In the next paragraphs, the devices and global variables will be described in detail, together with their graphic representation.

NOTE: In the diagnostic window only the devices enabled for the current access level are displayed.

5.2 Diagnostics window composition

It is possible to access the devices through the "Groups / Subgroups" structure, already described in the chapter System composition, which are then displayed in a tree structure.
At the head of the structure we find the group, symbolized by the icon:

 , followed by its Name and a Comment.

The structure can be expanded or collapsed by clicking on the ⊞ or ⊟ buttons. The tree can also be opened and closed by clicking on: **+**, **-**, left/right arrow key.

When a Group is opened, the following items are displayed in the tree:

- the "Devices List" of the Group, indicated by the icon 
- the Subgroups composing the Group, if any.

When a Subgroup is opened, the devices composing the subgroup are also displayed.

5.3 Representation of the Devices

The following information is shown with all the devices displayed.

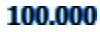
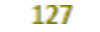
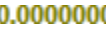
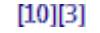
- a graphic symbol;
- its Status or current value;
- its Name;
- a Comment.

The list below contains the graphic representations of the devices, the type of device and the value displayed in real-time.

The status of digital inputs, digital outputs and flags is represented graphically by a LED which changes colour depending on whether the input is enabled or disabled.

In the case of Ports, that is a number of lines (8) represented at the same time, a row of LEDs will be shown, where the first line of the group is indicated by the right hand LED and the last one by the left hand LED.

Device	Symbol	Status	Real-time display
Digital input			status: Enabled = GREEN, Disabled = GREY
Digital output			status: Enabled = RED, Disabled = GREY
Analog input		22.000	current value

Device	Symbol	Status	Real-time display
Analog output			current numeric value in Volts
Input port			status of each line (as Digital input). Status: Enabled = GREEN, Disabled = GREY
Output port			status of each line (as Digital output). Status: Enabled = RED, Disabled = GREY
Axis			current absolute position
Timer			current value in seconds
Counter			current numeric value
Flag bit			status: Enabled = YELLOW, Disabled = GREY
Flag switch			status (as Flag bit). Status: Enabled = YELLOW, Disabled = GREY
Flag port			status of each line (as Flag bit). Status: Enabled = YELLOW, Disabled = GREY
Global variable			GPL code integer type global variable
Global variable			GPL code char type global variable
Global variable			GPL code float type global variable
Global variable			GPL code double type global variable
Global variable			GPL code string type global variable
Global variable			GPL code array type global variable
Global variable			GPL code matrix type global variable

5.4 Interacting with Devices

It is possible to interact with devices to read their status or modify their value, for diagnostic purposes. However, this is not possible for some types of devices, such as input devices and other devices protected by the Manufacturer. Should the Operator try to operate on these devices, a message will notify him.

When the device has been selected, double click on it with the mouse, or press **Enter**, or the **Space Bar**, to access the window that allows to change the status or the value of the device.

If the device concerned is a **Digital output** or a **Flag bit**, no window will appear, but the status of the device will be automatically changed. If the output is functioning correctly, the LED indicating its status will change colour.

If the device concerned is an **Output port**, point the mouse on the LED corresponding to the required output and *double click* on it to change its status.
The same applies to **Flag switch** and **Flag port**.

As far as **Analog outputs**, **Timers** and **Counters** are concerned, a dialog window is displayed, showing the current value and enabling to set immediately the new value we want applied to the device.

Axes interaction modes are described in the Manual Axes movement paragraph.

5.5 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
Ctrl+Alt+Shift and Enter	Shows the linearity corrector tables associated to an axis. If the key combination is activated, while an axis is selected in the device tree of a module, all the correctors associated to an axis appear, as if they were a matrix, in which the

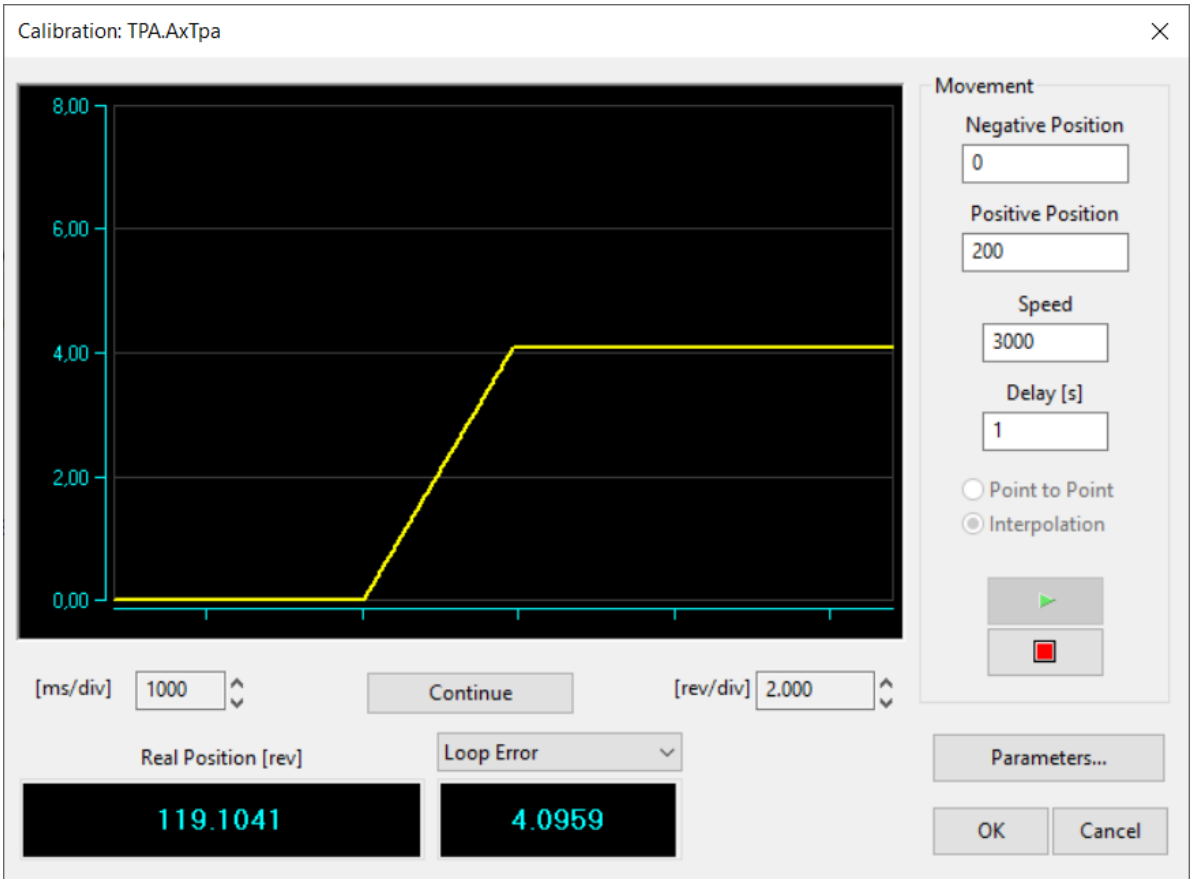
Key	Description
Ctrl+Alt+Shift and left click	columns are the associated axes (the first column is that of the self-correctors) and rows are the correction values. Any modified values are taken during the axis movement, but are not memorized on the disc.

5.6 Linearity correctors

In diagnostics, we can view and edit the linearity correctors of a selected axis by opening the context menu and selecting the option **Linearity correctors**. The option is only visible if some linearity correctors were defined for the axis in the configuration. As an alternative to the menu, we can use the key shortcut **[Ctrl+Shift+Enter]**.

5.7 Axis calibration control board

The axis calibration control board allows to modify axis configuration parameters and, at the same time, to move the axes and see its behaviour displayed on a virtual oscilloscope.
To access the axis calibration control board you need an access level to the main system or an access level as "Manufacturer". The calibration board is accessed in diagnostic or manual mode by double clicking on the axis to be calibrated while keeping the **[shift]** key pressed or from the context menu by selecting "Calibration".
The calibration control board shown in the following figure will be displayed:



To verify the axis behaviour as parameters change, the axis is moved continuously between two limit positions called **Positive Position** and **Negative Position**. As well as these parameters, axis movement **Speed** will also have to be set. In the early stages of calibration we suggest using a low speed value. A **Delay**, to be applied between movements, can also be fixed.

The oscilloscope window will display the axis loop error graph or one of the other axis values. It is possible, as with bench oscilloscopes, to scale the graph to adapt it to the size of the window and to examine it in detail. By means of a mouse or control keys and buttons you can examine the last calibration minute again, display one or two cursors to measure and check on the sampled data, enlarge an area of the graph to analyse the details of the sampled data, change the offset and the scale both in the x and y-coordinates. Moreover graph scrolling

can be interrupted by pressing the **Stop** button, to allow a careful study of the graph without having to stop the axis.
Besides the graph, two boxes showing (on the left) the real position and (on the right) the size displayed graphically. This can be set using the combo box situated above the display box.

To change the parameters of the axes, press the **[Parameters...]** button, which activates a window where most axis parameters can be edited directly. Once a change has been made to one or more parameters, it can be activated by pressing the **[Apply]** button.
This buttons allows seeing immediately the effect tha change has on the dynamic behaviour of the axis. If you press **[OK]** the changes remain in use; if you press **[Cancel]**, the values available before opening the **[Parameters...]** window are restored.

The main parameters to be operated on are the following:
- **Proportional** coefficient
- **Integral** coefficient
- **Derivative** coefficient
- **Feed Forward**: percentage of current speed provided directly by operation (independent of loop error)
- **Feed Forward Accel.**: percentage of speed reference provided directly by operation during axis acceleration and deceleration phases (in addition to feed forward)
- **Acceleration**: time of acceleration ramp
- **Deceleration**: time of deceleration ramp

The cursor in the graphical area

The cursor is a tool that lets us measure and view some trace data. It consists of a vertical line of the colour associated to the selected trace. You can move it within the chart by means of the cursor keys or by means of the mouse. In a tooltip associated with the vertical line you can display different values selectable from a popup menu that can be recalled by right-clicking next to the cursor.

In the popup menu you can choose the following options:

- **Channel**: it shows the list of the traces displayed in the chart and highlights by a visible check the one the cursor is referencing. Furthermore, you can select a new trace to be associate with the cursor.
- **Style**: it shows a list of data that can be displayed in the tooltip rectangle.
- **Value X-Y**: it shows the value on the x axis and on the y axis of the trace point on which the cursor is.
- **Value X**: it shows the value on the x axis of the trace point on which the cursor is.
- **Value Y**: it shows the value on the y axis of the trace point on which the cursor is.
- **Period**: it shows the value of the distance between two shown cursors, when measured along the X axis.
- **Peak-peak**: it shows the value of the distance between two shown cursors, when measure along the Y axis.
- **Frequency**: it shows the inverse of the distance along the X-axis.
- **Options**: it configures the display mode of the cursor and its associated tooltip.
- **Use Channel Color**: if enabled, the cursor is drawn in the colour of the sampled reference datum, otherwise the choice of the cursor colour is random.
- **Hide Hint On Release**: if enabled, the tooltip is shown only until the left mouse button is held, then it is hidden.
- **Flip Alignment**: the tooltip is normally displayed right of the cursor; if the option is enabled, the cursor is displayed left of the cursor.
- **Snap to Data-Point**: if enabled, it places the cursor only on sampled values.

Axis Calibration
Axis calibration is a delicate operation to be carried out with great care and caution.

Through the "CalibSampleTime" option in the [Albatros] unit in Tpa.ini, you can modify the data sampling time of an axis for the calibration window. The value in milliseconds and cannot be less than the frequency value of the control axis or less than 100.

Before calibrating the axes from the control board, set all the parameters in configuration and set the full-scale value for drive speed. The voltage value for the analog axes is 9 V which in Albatros corresponds to the maximum speed.

To avoid damaging the machine by setting incorrect parameters, it is advisable to set a low speed, for example the equivalent of 10% of axis maximum speed. This will avoid excessively violent reactions of the axes, even when the gain is set too high.

Normally, the machine is calibrated first for point to point movements and then for interpolation movements.

The first step, if it has not been done in configuration, is setting the acceleration and deceleration times. The longer the time, less will be the acceleration to which the axis is submitted.

The second step is setting a minimum gain, that allows axis movement. This is necessary to verify the correct drive calibration. Albatros is set to provide a reference of 9 volts when it reaches the maximum speed set in axis configuration. For example, if the axis is moved at a speed corresponding to 10% of maximum speed, and if the drive is calibrated correctly, the reference power should be 10% of maximum power, that is 0.9 volts. If this reference voltage is not obtained, the drive's full-scale value must be modified.

When the drive has been calibrated, we begin to increase the position loop gain, a little at a time and with great caution. Each time the position loop is increased, we must check that this has not caused conditions of excessive deflection or instability. In this phase, speed must be kept at 10% of maximum speed, or less, at all times. Moreover, it is always advisable to analyse the obtained speed profile carefully with the virtual oscilloscope, enlarging the image as much as possible to highlight the details.

When stable and ready axis performance has been obtained, the speed can be gradually increased. Check axis behaviour each time the speed is modified. The value of the gain must also be modified if it is not satisfactory. Gain and speed must never be increased abruptly, as apparently stable calibration conditions at a low speed may not be as stable at a higher speed.

When the optimal value of the Gain has been determined, if necessary, Integrative and Derivative coefficients and then the Feed Forward may be gradually increased to reduce the loop error, bringing it within acceptably precise values. The feed forward allows to eliminate the loop error almost completely during movement, but not during acceleration and deceleration. To further reduce the loop error in these phases, Feed Forward Acceleration can be increased. Normally, even very low values in this parameter are sufficient for satisfactory results.

As far as the axis calibration for interpolation movements is concerned, the same values set for point to point movement can be used, although the other axes of the machine must be taken into account. It is particularly important to balance the axis loop errors to obtain maximum precision during interpolation movements. This means that once the axis with the greatest loop error (at equal speed) has been identified, the calibration of the others must be adapted (limitedly to the interpolation parameters) to obtain identical loop errors.

6 Errors and Notifications

6.1 Introduction

Albatros manages error events and notifications

System errors
These are errors that Albatros system is able to automatically detect, both during program execution phases and during maintenance operations and plant diagnostics.

These include all kinds of errors, ranging from problems related to axis management to those that can arise during the program execution.

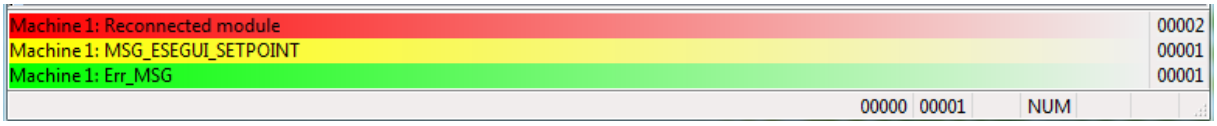
System errors can be managed inside the working programs, by means of the ONERRSYS instruction.
If this is not possible, program execution of the module where the error occurred is terminated. The explanation of each system error is described in the following pages of this manual.

Cycle errors
These are errors that occur during program execution, but that generally allow it to continue after removing the error itself. Cycle errors can be generated through GPL ERROR instruction.

Messages
These are warning messages that are generated in anomalous situations during program execution or notifications of help requests from the operator. However, they do not stop the execution of the program itself. Messages can be generated through GPL MESSAGE instruction.

Notifications
These are generic notifications not related to the machine loop and that are written by Albatros. They are not displayed in Albatros. The description of each notification is available in the following pages of the current manual.

Error Bar
The latest system error that occurred in chronological order is displayed in the error bar, together with the last cycle error and the last message.
System errors are indicated in red.
Cycle errors are indicated in yellow. These are errors occurring during program execution, which, however, usually allow it to continue after removing the error itself.
Messages are indicated in green.



Error Bar

The errors occurred from booting are displayed in a window opened either by double clicking on the *Error Bar* with the mouse or from the ***View*** menu. Additional information about system errors is also displayed in this window.

The window is divided into the following areas. In the **upper part** the following information is displayed:

- Type: identifies the type of error (system error, message and cycle error).
- Hour&Date: hour and date in which the error occurred.
- Description: description of the error.
- Code: number of error message.
- Task: name of the task generating the error (not present in Error Bar).

By *double clicking* on one of these columns, the information is sorted according to the content of the column.
The **bottom part** contains the cells:

- Cycle Errors: if this cell is enabled, cycle errors are also displayed.
- Messages: if this cell is enabled, messages are also displayed.
- All: if enabled, it lists all the messages, sent by any module of the system, concerning the type of information displayed.
- Module Name Cell: shows the name of the module whose information is being displayed. It also allows to select, in case of multi-module systems, the module whose information is required.

The control **buttons** are:

- **[Delete All]** clears from memory all the information displayed, without deleting it from the file.
- **[Delete]** clears current information from memory, without deleting it from the file.
- **[OK]** closes window.

Storage of errors and notifications in a report file
All errors are stored in a file for a historical reconstruction of the same. It is a text file in TSV format. The file name is MONTH (the current month number).TER. In the Albatros suite a ViewRER program is provided: it loads and displays the .TER files.

6.2 System Errors

6.2.1 Errors generated by axes control

1 AxisName: incorrect encoder connection

Cause:
The difference between the target position of a still axis, and the real position of the axis, exceeded 1024 encoder steps.
This often happens during axis commissioning, when the encoder phases are reversed. During normal functioning it occurs when an axis is moved manually, with the drive off, without setting the axis on FREE, or when the axis is subject to overshoot in the arrival phase, due to inaccurate calibration.
When this error occurs, the reference signal is set to zero and the axis is set on FREE.

Solution:
During axis commissioning, check the connection of the axis encoder phases (if necessary enable the encoder phases inversion option in axis configuration).
Verify axis calibration using the specific Diagnostic mode.

2 AxisName: not ended movement

Cause:
When the move has concluded, 5 seconds after the end of theoretical movement, the gap between the target position and the real position of the axis exceeds the window indicated in Configuration. This could be simply because the drive is off or disabled or it could be due to inaccurate drive offset regulation.
However, it could also be due to mechanical backlash on the axis or an excessively low axis position loop gain.

Solution:
Check that the drive is on and enabled. Verify axis calibration and adjust the drive offset of the concerned axis.

3 AxisName: servoerror

Cause
In any type of movement, the difference between the target position and the real position of the axis exceeded the maximum error indicated in Configuration or set with the SETMAXER instruction.
Normally this is due to the incorrect setting of the position loop gain or of the full-scale value of operation speed. It could also depend on excessive axis inertia.

Solution:
Verify the gain setting and the full-scale value of drive speed.
Check that the encoder and the motor/drive group are functioning correctly.
Check for any mechanical block.

4 AxisName: limit switch positive

Cause:
The target position of the axis exceeded the positive position limit indicated in Configuration or set using the SETLIMPOS instruction.

Solution:
Correct in the program the position exceeding the positive position limit or set new axis position limits.

5 AxisName: limit switch negative

Cause:
The target position of the axis exceeded the negative position limit indicated in Configuration or set using the SETLIMNEG instruction.

Solution:
Correct the position exceeding the negative position limit in the program or set new axis position limits.

10 AxisName: the Real-Time execution is faster than the profile construction

Cause
The real-time execution of the movement profile goes faster than the GPL generation of the profile itself. The lookahead is emptied faster than its filling. The error might be due to two generally simultaneous causes:
- the interpolation speed rate is too high with respect to the segment dimensions to be covered.
- the segments to be covered are too short.

Solution:
Verify that the interpolation rate speed set is not too high with respect to the segment dimensions to be covered; furthermore, verify that the interpolation segments to be covered are not too short.

6.2.2 Errors generated by remote I/O

2049 Receiver number: incorrect configuration

Cause:
The remote receiver received a different I/O expansions configuration from the one detected on the field. This can happen, if the remote is not equal to the one chosen in the hardware configuration of Albatros. For example, the remote receiver is an Albre16 and in Albatros a remote ALbre24 (GreenBus v3.0) or another TRS-IO with a wrong TRS-IO-E expansion number (GreenBus v4.0) has been configured.

Solution:
Verify Hardware configuration.

2050 Receiver number: disconnected

Cause:
The remote receiver does not respond to the transmitter's commands.

Solution:
Verify the receiver's power supply and the serial connection.

2051 Receiver number: reconnected

Cause:
The connection between the transmitter and the receiver has been restored.

2052 Receiver number: error reading Output not connected (number OutputNumber)

Cause:
The indicated digital output is in protection or in short circuit status, however, it is not in the status expected by control. The output is not associated to any logic device in Virtual-Physical Configuration, which indicates an incongruity between Configuration and the real cabling of the machine.

Solution:
Verify Virtual-Physical Configuration. Remove short circuit or verify that the applied load does not exceed maximum limits (consult technical documentation).

2054 Receiver number: wrong type

Cause:
During remote initialization, a different receiver from the one specified in configuration has been detected at a certain address.

Solution:
Verify that the Hardware Configuration agrees with the remote module setting.

2055 Receiver number: initialized

Cause:
The receiver has reconnected to the transmitter after an interruption caused by a power fail.

2056 Receiver number: +24 VDC power fail

Cause:
The field power (+24 VDC) of a I/O remote module is not active or is not correctly working.

Solution:
Check the +24 VDC power supply.

2057 GreenBus power fail

Cause:
The field bus power supply, connecting the I/O modules with the control, is not working properly. This power supply should have a nominal value of +12 Vcc and it is supplied by the control.

Solution:
Check the presence of the GreenBus power line, check the GreenBus cables. Switch-off and switch back on. If necessary, replace the control board.

2058 Receiver number: error reading DeviceType DeviceName

Cause:
The status of the specified output does not correspond to the set status. This could be due to a short-circuit, to overload protection or simply to the lack of power. The specified output can be a digital output, an analog output, an axis control output. The kind of output is specified in the error view.

Solution:
If it is a digital output, verify the +24V power supply (field side), remove the possible short circuit or the excessive output adsorption (see technical documentation). If it is an analog output or a axis control output, verify the presence and the value of the voltage set at the output (tester or oscilloscope), remove the possible short circuit or the excessive output adsorption (see technical documentation).

2059 Test failed on dual port memory of transmitter

Cause:
An error was generated during axis board initialization tests. Namely, initialization of the GreenBus transmitter (i296 microcontroller) failed. This could be due to the incorrect configuration of the board I/O and IRQ addresses or to conflict with other peripherals in the system. It could also be the consequence of a damaged axis board.

Solution:
Check the board configuration, check that there are no conflicts with other peripherals. If a remote module is used, retransmit the firmware to the module. Qualified technicians can test the Hardware of the i296 microcontroller dual port memory. If the problem persists, contact the manufacturer.

2060 Error initializing transmitter

Cause:
An error was generated during axis board initialization tests. Namely, firmware transmission to the GreenBus transmitter (i296 microcontroller) failed. This could be due to the incorrect configuration of the board I/O and IRQ addresses or to conflict with other peripherals in the system. It could also be the consequence of a damaged axis board.

Solution:
Check the board configuration, check that there are no conflicts with other peripherals. If a remote module is used, retransmit the firmware to the module. Qualified technicians can test the Hardware of the i296 microcontroller dual port memory. If the problem persists contact the manufacturer.

2061 Error transmitting firmware to transmitter

Cause:
An error was generated during axis board initialization tests. Namely, transmission of the remote I/O module configuration to the GreenBus transmitter (i296 microcontroller) failed.

Solution:
Verify hardware configuration, if a remote module is used, retransmit the firmware to the module. Qualified technicians can carry out a Hardware test on the i296 microcontroller RAM. If the problem persists contact the manufacturer.

2062 Error transmitting configuration to transmitter

Cause:
An error was generated during axis board initialization tests. In particular, remote I/O modules initialization failed.

Solution:
Verify the hardware configuration, if a remote module is used, retransmit the firmware to the module. Qualified technicians can carry out a Hardware test on the i296 microcontroller RAM. If the problem persists contact the manufacturer.

2063 Error transmitting configuration to receiver

Cause:
Error detected during initialization of a remote module.

Solution:
Verify the hardware configuration. Qualified technicians can carry out a Hardware test on the remote module. If the problem persists contact the technical support service.

2064 Receiver number: Incompatible firmware version

Cause:
Remote receiver has a firmware version, that is not compatible with the controller's firmware.

Solution:
Check installation of controller. If the problem persists, please contact the technical support service.

2065 Receiver number: Error in an asynchronous communication

Cause:
There was an error or a non-response during the communication of a command with the remote (GreenBus v.4.0).

Solution:
Check the connections and the GreenBus power supply. If the problem persists, please contact the technical support service.

2066 Receiver number: Generic error

Cause:
There was a generic error while communicating an event or an alarm from a remote (GreenBus v4.0)

Solution:
Check the connections and the GreenBus power supply. If the problem persists, please contact the technical support service.

2067 Receiver number: Error while transmitting the configuration

Cause:
A communication error while transmitting some configuration data to a remote (GreenBus v.4.0) occurred.

Solution:
Check the connections and the GreenBus power supply. Switch-off and switch back on. If the problem persists, please contact the technical support service.

2068 Receiver number: Internal error n. errornumber

Cause:
An internal error on the indicated remote has occurred.

Solution:
Please, contact the Manufacturer.

2069 Receiver number: +24 VDC power fail in bank number

Cause:
Field power supply (+24 VDC) of a output group connected to the same feeding clamp is not active or does not work correctly.

Solution:
Check the +24 VDC power supply.

6.2.3 Errors generated by MECHATROLINK-II

2308 Board BoardNumber: The initialisation failed due to an incorrect setting of a configuration parameter

Cause:
In virtual physical Configuration any axis (logical device) was not connected to the board with MECHATROLINK-II bus (physical device).

Solution:
Check the links in Virtual-physical Configuration.

2341 Board BoardNumber: The number of servodrives exceeds the maximum number allowed

Cause:
A number of servo-drives, exceeding the configuration set, was connected to a MECHATROLINK-II bus.

Solution:
Check the Axis Control Frequency value in system configuration.
In the table below the right values to be set according the number of servo-drives controlled by the board:

Board	Frequency Axis 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

2342 Board BoardNumber: The hardware address of servodrive Servo exceeds the maximum value allowed

Cause:
An axis (logical device), whose hardware address (physical device) is higher than the number of the servo-drives, that can be controlled by the board, has been connected to a MECHATROLINK-II bus.

Solution:
Check in system configuration the Axis Control Frequency value. In the table below the right values to be set according the number of servo-drives controlled by the board:

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

Check in Physical-Virtual Configuration the connection between logical device and physical device. For example, if the maximum number of servodrives is 8, so the connection between logical and physical device must included among the first 8 axes (from Ax1 to Ax8).

2349 Board BoardNumber: Servodrive Servo not connected

Cause:
Physical connection to the servo-drive of the indicated board is interrupted.

Solution:
Check servo-drive and MECHATROLINK-II bus cabling.

6.2.4 Errors generated by the CanBUS control

2761 Node number: disconnected

Cause:
The CAN node shown seems currently not to be plugged to field bus, that makes reference to the board shown, although it is included in the configuration.

2762 Node number: reconnected

Cause:
The CAN node shown seems to be just plugged to field bus, that makes reference to the board shown.

2763 Error: missing transmission

Cause:
Error inside the indicated board. Data transmission to the indicated Can node failed.

Solution:
Please, contact the Manufacturer.

2764 Node number: Error of non-reception

Cause:
Data reception from the CAN node failed.

Solution:
Check connection and power supply of the indicated CAN device. Check the cabling of the whole CAN line. Check line connection to the numeric control. Check coherence in the protocol settings of the indicated device CAN in respect of transmitter settings in the numeric control (baud rate, address, specific settings of the adopted protocol)

2765 Node number: Initialized

Cause:
The CAN node shown has been plugged to the field bus, and then it has been properly initialized.

2766 Fault condition on CAN interface

Cause:
An internal power supply failure of the CAN Interface device is reported.

Solution:
Please, contact the Manufacturer.

2767 CANopen status loss

Cause:
Because of a serious problem CAN transmitter is not operational anymore (Operational).

Solution:
Please, contact the Manufacturer.

2768 Node number: Error of PDO non-reception

Cause:
The numeric control did not receive the PDO, which was expected from CAN node.

Solution:
Check as follows:
- node supply.
- node has remained in pre-Operational mode.
- PDO and CAN bus data as configured in Albatros.

2769 Node number: Error receiving a non-configured node

Cause:
On the CAN network a node has been detected. It was not declared in the hardware configuration of Albatros.

Solution:
Check the node hardware address and the address of the node set in the hardware configuration.
Check that the node was declared in the hardware configuration, otherwise it is necessary to add it.

2770 Node number: Wrong configuration

Cause:
The RPDO and TPDO data description is wrong.

Solution:
Correct the PDO data description of transmission and reception in the hardware configuration.

2771 Node number: SDO communication error

Cause:
This CAN node did not respond in an asynchronous communication (SDO).

Solution:
Check the connection status of the node. If the problem persists, please, contact the Manufacturer.

2772 Timeout on querying nodes CAN cycle

Cause:
A timeout error on CAN cycle of node querying occurred

Solution:
In the hardware configuration change the value of the set sampling time.

3073 Node number: Emergency error n. ErrorNumber

Cause:
CANopen device has detected an error situation of the node, specified by the displayed code.
The error code is a hexadecimal number.
It is a matter of error situations pertaining to a single node and meeting with the standard CiA DS301-EMERGENCY protocol.

Solution:
Please, make reference to the node documentation.

3074 Node number: Generic CAN error n. ErrorNumber

Cause:
An internal error on the indicated node has occurred. The error code is a hexadecimal number.

Solution:
Please, contact the Manufacturer.

3088 CAN Board number: node NodeNumber: SDO communication error nr. ErrorNumber - description

Cause:
In a READDICTIONARY or WRITEDICTIONARY instruction, one or more requests for SDO read/write failed.
The failure of the instructions can be caused, for example, by the read request of a CANOpen object that is

not implemented in the device to which we are referring to. Or it can be related to the write, in a CANOpen register, of a data, that is not compatible with the object type (for example: writing attempt of a string into an object whose type is integer). The provided error code complies with the DS402 specifications and the textual description is also provided, in addition to the numerical code.
The error code is a hexadecimal number.

Solution:
Check the parameters correctness of BaudRate, Sampling time, etc., set in the hardware configuration and the parameters of possible READDICTIONARY and/or WRITEDICTIONARY instructions which are in the GPL code.

6.2.5 Errors generated by bus EtherCAT control

3329 Error in the communication socket initialization

Cause:
The firmware could not communicate with the network board.

Solution:
If the board has been configured in a RTX System, check that the .ini files existing in the Albatros FW subfolder are properly written. To check the syntax of the files, refer to the RTX installation manual in Albatros (InstallationRTXGuide.pdf).

3330 Error during the EtherCAT network scan

Cause:
While pre-scanning the EtherCAT network, the master did not receive any answer from some or from all the configured slaves or the configuration does not match with the real EtherCAT network available in the field.

Solution:
Check the cabling between the EtherCAT master and the slave.
Check the descriptions of the devices in the hardware configuration. Hardware Diagnostic window can help to find the error. Here, the existing nodes are displayed and, if wrongly configured, besides the device name found, the name of the expected device is displayed.

3331 Error in the configuration of the transmission mailbox

Cause:
The EtherCAT node has not responded to the command given by the Master. Potential causes: absent communication, faulty node...

Solution:
Check the cabling and the remote operation.

3332 Error in the configuration of the receive mailbox

Cause:
The EtherCAT node did not respond to the command given by the Master. Possible causes: absent communication, node failure...

Solution:
Check the cabling and the remote operation.

3333 EtherCAT board number: Error in the expansion type of node NodeNumber

Cause:
The type of expansions configured on an EtherCAT node in the hardware configuration does not correspond to the kind of the expansions actually present. (For example, in the hardware configuration a TRS-CAT has been defined with a TRS-IO-E expansion, while in the system a TRS-CAT is available with a TRS-AN-E expansion).

Solution:
Check that the devices set in the hardware configuration correspond to those available.

3334 Error during the PDO configuration

Cause:
The EtherCAT node, for which you tried to configure the PDOs, is not available on the network or has a failure.

Solution:
Check that the EtherCAT network configuration, described in the Albatros configuration, corresponds to the network physical configuration.

3335 Node NodeNumber in alarm (ErrorNumber)

Cause:
The indicated node is in an alarm situation.

Solution:
Check the alarm code in the following table

Alarm code	Description
0x0001	Unspecified error
0x0002	No memory
0x0011	Invalid requested status change
0x0012	Unknown requested status
0x0013	Bootstrap not supported
0x0014	No valid firmware
0x0015	Invalid mailbox configuration
0x0016	Invalid mailbox configuration
0x0017	Invalid sync manager configuration
0x0018	No valid inputs available
0x0019	No valid outputs
0x001A	Synchronization error
0x001B	Sync manager watchdog
0x001C	Invalid Sync Manager Types
0x001D	Invalid Output Configuration
0x001E	Invalid Input Configuration
0x001F	Invalid Watchdog Configuration
0x0020	Slave needs cold start
0x0021	Slave needs INIT
0x0022	Slave needs PREOP
0x0023	Slave needs SAFEOP
0x0024	Invalid input mapping
0x0025	Invalid output mapping
0x0026	Inconsistent settings
0x0027	Free-Run not supported
0x0028	Synchronization not supported
0x0029	Free-Run needs 3 buffer mode
0x002A	Background watchdog
0x002B	No valid inputs and outputs
0x002C	Fatal Sync error

0x002D	No Sync error
0x0030	Invalid DC SYNCH Configuration
0x0031	Invalid DC Latch Configuration
0x0032	PLL Error
0x0033	Invalid DC IO Error
0x0034	Invalid DC Timeout Error
0x0035	DC Invalid Sync Cycle Time
0x0036	DC Sync0 Cycle Time
0x0037	DC Sync1 Cycle Time
0x0041	MBX_AOE
0x0042	MBX_EOE
0x0043	MBX_COE
0x0044	MBX_FOE
0x0045	MBX_SOE
0x004F	MBX_VOE
0x0050	EEPROM no access
0x0051	EEPROM error
0x0060	Slave restarted locally

3336 EtherCAT board number: The expansion number of node NodeNumber is wrong

Cause:
The configured expansion number on an EtherCAT node in Albatros does not correspond to the number of the present expansions. (For example, in the hardware configuration a TRS-CAT with two TRS-IO-E expansions has been defined, while only one expansion is available in the system).

Solution:
Check that the devices described in the hardware configuration correspond to those available.

3337 EtherCAT board: Node NodeNumber: Disconnected

Cause:
The EtherCAT node has not returned any response to the control. The node could be disconnected from the network or it could be off.

Solution:
Check the cabling and the power supply to the node.

3338 EtherCAT board: Node NodeNumber: Reconnected

Cause:
The EtherCAT node indicated has been reconnected to the network and has started again to respond to the control requests.

3340 EtherCAT board: Node NodeNumber did not respond to the request (Code)

Cause:
The EtherCAT node, queried using the SDO service, did not respond to the request. Furthermore, if the device has provided an abort code, this is reported in the error message shown (code). This error may appear during the execution of the SETPZERO, SETPFLY, and FASTREAD instructions on EtherCAT axes, since they use SDO communication for drive configuration. In this specific case, in addition to the standard abort codes, the error message can contain the following codes:

- 1 Internal Timeout expired: the node did not respond within 250 ms.
- 00000005: internal index of incorrect mailbox.
- 00000006: data received not compliant with the request.

Solution:
Contact the manufacturer.

3341 EtherCAT board: Node NodeNumber does not exist

Cause:
A not-existing node was asked to perform a command.

Solution:
Check the node number in the GPL instruction, that generated the error.
Connect the node.

3342 Disconnected cable

Cause:
The EtherCAT cable is not connected to the control.

Solution:
Check that the cable is correctly connected to the control and that it is not damaged.

3343 EtherCAT board number: Node NodeNumber does not switch to the SAFE-OPERATIONAL status (Code)

Cause:
The indicated EtherCAT node did not switch to the SAFE-OPERATIONAL status.

Solution:
Contact the manufacturer and report the indicated error number (code), which represents the ALstatuscode code.

3344 EtherCAT board number: Node NodeNumber does not switch to the OPERATIONAL status (Code)

Cause:
The indicated EtherCAT node did not switch to the OPERATIONAL status.

Solution:
Contact the manufacturer and report the indicated error number (code), which represents the ALstatuscode code.

3345 EtherCAT board: Unstable communication

Cause:
The communication on the EtherCAT network is unstable due either to interferences or cables or faulty nodes.

4400 Too many active axes in FASTREAD (Function:NameFunction line:NumberLine)

Cause:
There was an attampt to use a FASTREAD instruction with a number of axes higher than the number allowed. It is necessary to keep in mind that the axes equipped with additional encoder correspond to a double axis (please see the instruction SWITCHENC).

Solution:
Reduce the number of axes used with additional encoder.

6.2.6 Errors generated by initialization

769 Error in software configuration

Cause:
The hardware configuration of the remote module does not match the software configuration specified in the system configuration.

Solution:
Please, check that the hardware parameters of the remote module and the software parameters are congruent to each other.

770 Wrong IRQ number in configuration

Cause:
The IRQ of the axis board has not been set correctly in the Module configuration. Normally a hardware conflict with other peripherals in the system is the cause.

Solution:
Verify in the motherboard BIOS settings that the IRQ used by the axis board is reserved for "Legacy ISA" only. Verify that no other peripherals are using the same IRQ assigned to the axis board. If possible, modify the IRQ of the peripheral in conflict with the axis board, otherwise modify the axis board IRQ.

772 Error reading the buffer memory area while initialising

Cause:
An error was generated during axis board initialization tests. Namely, the buffered RAM (Dallas) test failed. This could be due to the incorrect configuration of the board I/O and IRQ addresses or to conflict with other peripherals in the system. It could also be the consequence of a damaged axis board.

Solution:
Verify the hardware configuration. Qualified technicians may carry out a Hardware test on the i296 microcontroller RAM. Notice that the RAM Hardware test implies clearing all the data saved in it. The buffered RAM contains the values of certain devices, such as counters, timers and axis DAC offsets.
Save these values before running the test.
If the problem persists, please contact the manufacturer.

773 Reached maximum number of axes in configuration

Cause:
An attempt to configure more axes than allowed.

Solution:
Reduce the number of axis to be configured. For further information, please contact the Manufacturer.

774 Axis Real-Time is not running

Cause:
The axes management firmware was initialized but is not functioning properly. Normally, a hardware conflict with other peripherals in the system is the cause.

Solution:
Check that there is no conflict with other peripherals. Change the configuration of the peripherals causing the conflict or remove these peripherals from the system.

775 No time left to run GPL

Cause:
The execution of a real-time task takes up too much cycle time. This is generated when a real-time task does not end before the start of the next axis real-time task (for example when an infinite cycle has been created).

Solution:
Change the GPL code so as to reduce the length of the real-time task.

776 Real-Time execution time too long

Cause:
The execution of a real-time task takes up too much cycle time. The execution time is slightly over the maximum allowed.

Solution:
Change the GPL code so as to reduce the length of the real-time task.

777 Watchdog expired

Cause:
The firmware is stuck.

Solution:
Please, contact the Manufacturer.

778 Main firmware code is blocked

Cause:
The firmware has crashed for more then 5 real-times.

Solution:
Please, contact the Manufacturer.

1025 Board BoardNumber: It does not respond to command number

Cause:
An axis board was detected during initialization, but it does not respond correctly to commands.

Solution:
Qualified technicians can carry out a Hardware test on the axis board. If the problem persists, please contact the constructor.

1026 Board BoardNumber: Error transmitting firmware to the axis board

Cause:
Impossible to send the firmware to the board.

Solution:
Contact the manufacturer.

1028 Board BoardNumber: Firmware not present

Cause:
The firmwares on board are not correct for the type of board detected.

Solution:
Transmit the correct firmware.

1029 Board BoardNumber: Main blocked

Cause:
The firmware of the board did not start to run.

Solution:
Contact the manufacturer.

1031 Board BoardNumber: Initialization error

Cause:
An error was generated during initialization process of axis board.

Solution:

Check and fix the causes of the system errors occurred in the moments before the occurrence of the current error. Then initialise the system.

1032 Board BoardNumber: Dual port memory test failed

Cause:
An error occurred while testing in the initialization phase of the axes board. More particularly, the initialisation of the dual port memory failed.
This is typically due to a hardware conflict with other peripheral devices present in the system, but it could also depend on a damaged board.

Solution:
Check the board configuration, check that there are no conflicts with other peripheral devices. If a remote module is used, retransmit the firmware to the module. Qualified technicians can carry out a Hardware test on the dual port memory. If the problem persists, please contact the Manufacturer.

1033 Board BoardNumber: Firmware Boot code is not running

Cause:
The booting firmware was initialized but is not working properly. This is typically due to a hardware conflict with other peripheral devices in the system.

Solution:
Check the board configuration, check that there are no conflicts with other peripheral devices. If a remote module is used, retransmit the firmware to the module. Qualified technicians can carry out a Hardware test on the dual port memory. If the problem persists, please contact the Manufacturer.

1035 Board BoardNumber: Not present

Cause:
An error occurred while testing in the initialization of the axes board. More particularly, the board was not detected.

Solution:
Check that the board is actually in the system and that it is not damaged. Qualified technicians can perform a Hardware test of the board. If the problem persists, please contact the Manufacturer.

1037 Board BoardNumber: Failed to open the dual port memory

Cause:
Failed to open the dual port memory board.

Solution:
Qualified technicians can perform a Hardware test on the board. Please, contact the manufacturer.

1039 Board BoardNumber: Watchdog expired

Cause:
The firmware of the *BoardNumber* axes board is blocked.

Solution:
Contact the Manufacturer.

1040 Board BoardNumber: +24 VDC power fail

Cause:
There is not any field power supply (+24 VDC) of the outputs or it is not working properly.

Solution:
Check the +24 VDC field power.

1047 Board BoardNumber: Software configuration not allowed

Cause:
The device has received a configuration that is not compatible with the hardware in use or enabled. For instance, the configuration of an axis that is not enabled or not present on the device is required.

Solution:
Check that the board hardware parameters correspond to the software ones.

1052 Board BoardNumber: Boot code is running

Cause:
The board is in set in Safe mode and it is running the boot code.

Solution:
Please contact the Manufacturer.

1053 Board BoardNumber: Axis Watchdog expired

Cause:
A serious error while executing the firmware of the axis control board occurred. Axes are disabled and the SYSOK signal, if any, is lowered. Do not reset the system.

Solution:
Please contact the Manufacturer.

1055 Watchdog expired for board BoardNumber

Cause:
The firmware of the board BoardNumber is blocked.

Solution:
Contact the Manufacturer.

1056 Board BoardNumber: CAN interface power failed

Cause:
Power supply of the transmission device on CanBus line in the indicated board failed. It can depends on a short circuit, on a bus cabling error or on a damaged board.

Solution:
Check the cabling of the whole CAN line. Check the line connection to the numeric control. Remove the presence of any short circuit. If communication is not restored, contact the manufacturer.

1057 Board BoardNumber: Internal error n° ErrorNumber

Cause:
Error in the hardware of the node.

Solution:
Please contact the Manufacturer.

6.2.7 Errors generated by memory management

1281 Error in the memory allocation on the heap area

Cause:
Available RAM memory is not sufficient to satisfy the requirement, for example, of a global matrix.

Solution:
Reduce the size of the global variables allocated in RAM.

1286 Error handling heap

Cause:
Error in the firmware's memory handling.

Solution:

Please contact the Manufacturer.

1287 Too many memory deallocations from the heap

Cause:
Firmware made an error while managing the memory.

Solution:
Please contact the Manufacturer.

1289 Error creating global variables

Cause:
Too many global variables were defined, or the defined global matrixes are too large.

Solution:
Reduce the number of global variables or the size of the matrixes.

1290 Error in the dimension of non-volatile variables

Cause:
Too many non volatile variables were defined, or the defined non volatile matrixes are too large.

Solution:
Reduce the number of non volatile variables or the size of the non volatile matrixes.

1291 Error in the dimension of read-only variables

Cause:
Too many read only variables were defined, or the defined read only matrixes are too large.

Solution:
Reduce the number of read only variables or the size of the read only matrixes.

6.2.8 Errors generated by faults

1559 Breakpoint Trace

Cause:
Serious firmware error.

Solution:
Please contact the Manufacturer.

1569 Invalid microprocessor operating code

Cause:
The microprocessor has encountered an unknown instruction. This could either be due to PC hardware problems or the files containing Albatros firmware could be damaged.

Solution:
In the case of a local module, check that the files are not damaged and try reinstalling Albatros. In the case of Clipper modules, update the firmware. Run a PC hardware test, especially on the RAM. If the problem persists, please, contact the Manufacturer.

1586 INTEGER value divided by zero

Cause:
An attempt to divide an INTEGER by zero.

Solution:
Please check that all the divisions in the GPL functions are correct.

1600 Overflow in the result of a floating point operation

Cause:
The result of an operation between FLOATs is greater than the capacity of the recipient:
± 3,402823E+38 for floats
± 1,79769313486231E+308 for doubles.

Solution:
Please check that the floating point calculations in the GPL functions are correct.

1601 Underflow in the result of a floating point operation

Cause:
The result of an operation between FLOATs is smaller than the capacity of the recipient:
± 1,401298E-45 for floats
± 4,94065645841247E-324 for doubles.

Solution:
Please check that the float calculations in the GPL functions are correct.

1602 Invalid argument in a floating point operation

Cause:
An operand different from float type was used in a float operation.

Solution:
Please check that float calculations in the GPL functions are correct.

1603 Floating point value divided by zero

Cause:
An attempt to divide a float or double by zero. Raised also when a logarithm of zero is executed.

Solution:
Please check that all the divisions in the GPL functions are correct.

1604 Incorrect result in a floating point operation

Cause:
The result of an operation between floats is incorrect.

Solution:
Please check that the float calculations in the GPL functions are correct.

1605 Incorrect value for a floating point data

Cause:
The use of a smaller floating point value than the minimum representable value:
± 1,401298E-45 for floats
± 4,94065645841247E-324 for doubles.

Solution:
Please check that float calculations in the GPL functions are correct.

1728 Attempt to get access to an invalid address

Cause:
The program accessed an invalid memory area.

Solution:
Please check the global/local variable congruity. If the problem persists, please report the anomaly.

1735 Generic exception

Cause:
An unknown exception occurred.

Solution:
Please contact the Manufacturer.

1736 Data not aligned

Cause:
Serious error of the firmware.

Solution:
Please contact the Manufacturer.

1801 Temperature alarm

Cause:
Temperature of controller's CPU has exceeded maximum permissible limits.

Solution:
Please make sure there is no ventilation problem or anything causing overheating. If the problem persists, please contact the technical support service.

1802 Fan alarm

Cause:
CPU fan of the control is not working properly. Problem can lead to CPU overheating in short time.

Solution:
Please contact the Manufacturer.

1803 Unstable CPU frequency

Cause:
CPU work frequency is not stable.

Solution:
Please contact the Manufacturer.

6.2.9 Errors generated by GPL instructions

4097 The DeviceType device DeviceName is not configured

Cause:
A GPL instruction used a non-configured device, that is a device with no Virtual-Physical connection. It can be generated by all the instructions to which a device is passed as parameter.

Solution:
Please check in the control configurations that all the devices used by the functions have a Virtual-Physical connection.
Then retransmit configurations to the board.

4098 The global variable VariableName does not exist

Cause:
A GPL instruction received an undefined global variable as argument.
This usually happens when the control was not correctly initialized.

Solution:
Please recompile the whole GPL code and initialize control again.

4099 Function FunctionName not found

Cause:
An absent function was called.
It can occur when the control has not been initialized after modifying the GPL code.

Solution:
Please recompile the whole GPL code and initialize control again.

4101 Inconsistent management of axis AxisName

Cause:
An illegal status change was performed on an axis. For status changes consult the relative documentation. The error could be generated by any of the instructions managing the axes, normally it occurs in the following cases:

- if an attempt is made to interpolate, coordinate an axis already occupied in a point-to-point movement (or vice versa).
- if a Chain, SetPFly or SetPZero instruction is executed on an axis in transparent mode.
- if an attempt is made to interpolate, coordinate a slave axis.

Solution:
Please check that all axis transfers end with a wait in position instruction, especially if the axes alternate different types of movements (point-to-point, interpolation, etc.).

4105 Instruction not executable on axis AxisName

Cause:
An attempt to execute an instruction on an axis which does not support it. For example, an interpolation instruction on a stepper axis.

Solution:
Please correct the GPL code.

4106 The remote module of the stepper axis AxisName is not connected

Cause:
An attempt to operate on a stepper axis that is not connected to the control.

Solution:
Please check the connection of the remote controlling the axis.

4107 SYSOK instruction has incorrect arguments

Cause:
A SYSOK instruction with incorrect arguments was executed. Verify whether one or more digital outputs passed as instruction arguments are not correctly configured.

Solution:
Please check the GPL code and the Virtual-Physical configuration.

4108 AxisName: Final position beyond software limits

Cause:
An attempt to move an axis beyond the limits set in configuration or by the GPL code.

Solution:
Please correct the machining program that caused the error. If necessary, correct the GPL code or axis configuration.

4110 Wrong speed

Cause:
An axis was assigned a null or negative speed.

Solution:

Please correct GPL code.

4111 Negative Acceleration on axis AxisName

Cause:
An axis was assigned negative acceleration.

Solution:
Please correct GPL code.

4112 Negative Deceleration on axis AxisName

Cause:
An axis was assigned negative deceleration.

Solution:
Please correct the GPL code.

4114 Axis AxisName: reset on Fast Input not effected

Cause:
The Fast Input Reset (on the fly homing) was not completed correctly. This procedure enables to reset to zero the position of a moving axis, the moment the corresponding fast input changes status. If the axis concludes the movement in process with no input switching, the system error is generated. This could be due to the incorrect setting of axis movement parameters or to a cabling problem in the fast input.

Solution:
Please check the GPL code implementing on the fly homing, check fast input cabling.

4115 Axis AxisName: zero pulse not found

Cause:
The encoder zero pulse reset was not completed correctly. This procedure enables to reset to zero the position of a moving axis the moment the encoder 's zero pulse is detected. If the axis reaches the pulse search position without detecting the zero pulse, the system error is generated. This could be due to the incorrect setting of axis movement parameters or to a cabling problem in the pulse signal (axis connector C phase).

Solution:
Please check the GPL code implementing pulse homing, check axis cabling.

4353 Unknown instruction code (Function:FunctionName line:LineNumber)

Cause:
An illegal instruction was detected during the execution of a GPL function. Generally this indicates that the files containing the compiled GPL code are damaged. Verify also whether the control software and firmware were updated without recompiling the GPL code, as the earlier version could contain instructions which are no longer supported by the new one.

Solution:
Recompile the whole GPL code and initialize control. If the problem persists, please contact the Manufacturer.

4354 Incorrect mathematical operation (Function:FunctionName line:LineNumber)

Cause:
A GPL instruction tried executing an incorrect mathematical operation, such as dividing by zero. Or data introduced in the GPL instruction is incongruent. This error is often generated by interpolation movement instructions, as this is the Firmware that performs the most mathematical operations.

Solution:
Check that the data passed to interpolation instructions is correct. If the problem persists, please report the problem to the Manufacturer.

4355 Incorrect address of matrix or vector (Function:FunctionName line:LineNumber)

Cause:
A GPL instruction tried accessing an array or matrix element exceeding maximum size. For example, it tried accessing element 10 of a 5-element array.
It could be generated by any instruction accepting an array or matrix as a parameter.

Solution:
Please check that all the matrix and array indexes passed to the instructions are within the array and matrix size.

4356 Instruction RET without CALL (Function: FunctionName line: LineNumber)

Cause:
A RET instruction was executed although the stack did not contain the relative return address. Declaring a sub-procedure before the exit function FRET instruction, without protecting it with a GOTO to avoid accidental execution, is the most frequent cause. It is also possible that an accidental jump occurred in a sub-procedure.

Solution:
Please check the GPL program flow. When possible, place sub-procedures at the end of the body of the function (after the FRET instruction).

4357 Local variable does not exist (Function:FunctionName line:LineNumber)

Cause:
A GPL instruction tried to access a local variable which was not allocated.

Solution:
Recompile and retransmit all board functions. If the problem persists, please report the problem.

4358 Jump label does not exist (Function: FunctionName line: LineNumber)

Cause:
A GPL instruction jumped to a non-existing jump label.
It can be generated by GOTO, CALL, FCALL and all IFs.

Solution:
Recompile and retransmit all the functions to the board. If the problem persists, please report the problem.

4359 Incorrect macro argument (Function:FunctionName line:LineNumber)

Cause:
A GPL instruction was passed invalid arguments. It can be generated by any instruction. However, in the great majority of cases, the GPL system tries to correct the situation automatically, by performing automatic type conversions (cast), which may imply wasting time. The error is generated when these conversions are not possible and especially in the following cases:

- instructions operating on specific devices (SETTIMER, SETCOUNTER) that are given a different type of device.
- instructions operating on bits that are given a floating point number (AND, OR, etc)
- instructions operating on matrixes or arrays that are given a simple variable (SORT, MOVEMAT, etc.)
- instructions that operate on strings that are not given strings.

The error is generated even when the system tries to carry out an instruction in a board that does not manage such an instruction (for example an instruction SENDPDO or an instruction RECEIVEPDO in a board that is not a TMSCan or a TMSCan+ board)

Solution:
Please correct the GPL code.

4360 Error in the memory allocation during the execution (Function:FunctionName line:LineNumber)

Cause:
The GPL function tried to allocate a region of memory for internal use, but did not find available memory. The error could indicate a temporary situation, due, for example to an excessive number of tasks in execution at the same time or to excessively large global variables.

Solution:
Please check the size of the global and local variables and if possible reduce their size. Check if too many tasks are in execution at the same time and if necessary reduce them.

4361 Too many tasks enabled (Function:FunctionName line:LineNumber)

Cause:
An attempt to execute more than 256 tasks at the same time.

Solution:
Please reduce the number of tasks in execution at the same time.

4362 Incorrect matrix format (Function:FunctionName line:LineNumber)

Cause:
An instruction operating on matrixes has found an invalid format. The instructions that could generate this error are the following:
- MOVEMAT if the format of the source matrix and the destination matrix do not correspond.
- CLEAR if a non-existing row of the matrix is being deleted.
- GETAXIS if the format of the matrix, passed as a parameter, does not correspond to the format expected by the instruction (consult GPL language documentation).

Solution:
Check the above mentioned instructions in the task that generated the error. Check especially that the matrixes passed to MOVEMAT have the same number of columns of the same type and that the matrix passed to GETAXIS has the right format.

4363 Too many active ONINPUT instructions (Function:FunctionName line:LineNumber)

Cause:
More than 128 OnINput instructions have been activated.

Solution:
Please reduce the number of ONINPUT.

4364 Axis already engaged with local reference (Function:FunctionName line:LineNumber)

Cause:
The error concerns the activation of the rototraslate axis terns to execute interpolations on a number of Cartesian axes.
There was an attempt to execute a SETRIFLOC passing to the instruction an axis that was already engaged in a reference axis tern. It can also be generated if a RESRIFLOC is executed on an axis which is not engaged in any axis tern. It is also possible that no reference terns were available (there can be a maximum of 32 terms).

Solution:
Check that the terns passed by the SETRIFLOC have no axes in common.
Check RESRIFLOCs.
Check that the RESRIFLOC is preceded by wait in position instructions.
Remember that the RESRIFLOC is not executed until the interpolation has concluded.

4365 Instruction ONINPUT activated on the same INPUT (Function:FunctionName line:LineNumber)

Cause:
The same input was passed to an ONINPUT instruction more than once.

Solution:
Please check that the same input is not sent as a parameter to two ONINPUTs.

4366 Too many ONFLAG instructions active (Function:FunctionName line:LineNumber)

Cause:
More than 128 OnFlags instructions were activated.

Solution:
Please reduce the number of ONFLAGs.

4367 Instruction ONFLAG activated on the same FLAG (Function:FunctionName line:LineNumber)

Cause:
An ONFLAG instruction was passed to the same flag more than once.

Solution:
Please check that the same flag is not passed as a parameter to two ONFLAGs.

4368 A ReadOnly variable writing has been attempted (Function:FunctionName line:LineNumber)

Cause:
An attempt to write on a readonly variable.
Readonly variables are always global and reside in the command flash. They are indicated as "static" in the global variables editor. If an attempt is made to write on one of these global variables this system error is generated.
The error is also generated if variables residing in the buffered RAM ("non volatile") are used as arguments of certain write instructions.
For instance, in the instruction COORDIN the variable passed to show the row being processed must be in RAM.

Solution:
Please check all the static and non volatile variables.

4369 Too many master axes active (Function:FunctionName line:LineNumber)

Cause:
An attempt to activate more than four axes as master at the same time.
This error is only generated while executing the CHAIN instruction.

Solution:
Please reduce the number of master axes.

4370 Too many slave axes active (Function:FunctionName line:LineNumber)

Cause:
An attempt to activate more than eight axes as slaves of a single master axis.
This error is only generated while executing the CHAIN instruction.

Solution:
Please reduce the number of slave axes.

4372 Incorrect use of an instruction (Function:FunctionName line:LineNumber)

Cause:
This error is generated in one of the following situations:
1. You are using an instruction to manage a mailbox (sendmail, waitmail, endmail, ifmail) or an instruction for IPC management (sendipc, testipc, waitipc) within a function called by an Errsys, OnInput or OnFlag instruction.
2. You are using an IfError or an IfMessage instruction without having enabled status alarm management.
3. You are using the Watchdog instruction without the presence of the TMSWD board.
4. The parameters defined in an interpolation instruction (linearinc, linearabs, circle, circinc, circabs, helicinc and helicabs) are not consistent. For example, the number of declared axes is different from the number of declared positions or a greater number of axes has been declared than that the instruction can manage.

Solution:
Solutions for each listed cause:
1. Move to another function the instruction that causes the error or remove the instruction.
2. Check that the status alarm management has been enabled. In the tpa.ini file in the [ALBATROS] section under the AlarmsHaveStatus the assigned value must be 1.
3. Remove the WatchDog instruction or provide a TMSWD board.
4. Check that the parameters of the GPL instruction are correct. Each axis must correspond to a position. The number of the axes declared in the instruction should not be greater than the number of the axes that the function can manage. For example, the LINEARABS instruction can manage up to 6 axes. If more than 6 parameters are declared, the instruction generates the system error.

4373 Can't read feed rate (Function:FunctionName line:LineNumber)

Cause:
The GETFEED instruction was used on a TmsBus or TmsCan board that is not a master.

Solution:
in the hardware configuration please check that the board on which the feedrate is connected is master.

4374 Too many IPC instructions in execution (Function:FunctionName line:LineNumber)

Cause:
The maximum limit of 16 IPC instruction in execution at the same time was exceeded.

Solution:
Please modify the GPL code.

4375 FASTREAD executed on axes from different boards (Function:FunctionName line:LineNumber)

Cause:
A FASTREAD instruction was executed, although the axes passed as parameters were not all connected to the same board.

Solution:
Please modify the GPL code or the Virtual-Physical configuration as required.

4378 Instruction not enabled (Function:FunctionName line:LineNumber)

Cause:
An attempt to use an instruction whose execution was not enabled. Probably, the hardware key is not correctly inserted or is missing.

Solution:
Please insert the hardware key correctly. If the problem persists, contact the manufacturer.

4379 The instruction cannot be used in functions launched by Interrupt (Function:FunctionName line:LineNumber)

Cause:
An attempt to use an illegal instruction in a function launched by interrupt. The functions launched by interrupt are passed as parameters to ONERRSYS, ONINPUT and ONFLAG instructions.

Solution:
Modify GPL code. Please consult the list of instructions which can not be used with interrupt.

4380 Too many writing requests into buffer memory area (Function:FunctionName line:LineNumber)

Cause:
Too many write operations were performed on the buffered memory at the same time (buffered memory is characterised by relatively slow access).

Solution:
Please verify the instructions that perform write operations on the variables allocated in the buffered memory: counters, timers, matrixes and variables declared as "non volatile".

4381 Cannot use a serial channel not yet open (Function:FunctionName line:LineNumber)

Cause:
An attempt to execute an instruction that operates on the serial port, before executing the COMOPEN instruction for this port.

Solution:
Please modify GPL code.

4382 Cannot open a serial channel already open (Function:FunctionName line:LineNumber)

Cause:
A COMOPEN instruction was executed on a serial port that has already been opened with the same instruction.

Solution:
Please modify GPL code.

4383 Attempt to open too many auxiliary processes (Function:FunctionName line:LineNumber)

Cause:
An attempt to open more than 4 auxiliary processes at the same time.

Solution:
Please modify GPL code.

4384 Auxiliary process not in execution (Function:FunctionName line:LineNumber)

Causa:
An attempt to access an auxiliary process which is not in execution.

Solution:
Please modify GPL code.

4385 Attempt to open an auxiliary process from another task (Function:FunctionName line:LineNumber)

Cause:
An attempt to open an auxiliary process from a different task from the one that started execution. Auxiliary tasks can only be used by the tasks that started their execution.

Solution:
Please modify GPL code.

4391 Error activating SYSOK (Function:FunctionName line:LineNumber)

Cause:
The SYSOK signal activation was not successfully concluded. This is often due to a malfunctioning Greenbus transmitter on the axis board.

Solution:
Qualified technicians can perform a Hardware test on the i296 microcontroller dual port memory. If the problem persists, please contact the constructor.

4394 Too many cycle errors (Function:FunctionName line:LineNumber)

Cause:
There are more than 2000 cycle errors active.

Solution:
Please correct GPL code by limiting the number of notifications.

4395 Too many messages (Function:FunctionName line:LineNumber)

Cause:
There are ore than 2000 messages active.

Solution:
Please correct GPL code by limiting the number of notifications.

4397 Stack overflow (Function:FunctionName line:LineNumber)

Cause:
A GPL function stack exceeded the maximum limit of 2Kbyte.

Solution:
Compile the GPL code again and check in the compiler report the estimated stack space of the function that generated the system error. Then reduce the number of local variables and of parameters passed to the functions (replacing them, for example with global variables). Please reduce the number of CALLs.

4398 Stack underflow (Function:FunctionName line:LineNumber)

Cause:
It can only occur in case of a serious Firmware error, such as the incorrect management of function parameters or local variables.

Solution:
Please contact the Manufacturer.

4399 Parameter out of range (Function:FunctionName line:LineNumber)

Cause:
A GPL variable or a device was assigned a value outside the allowed range.

Solution:
Please correct and compile the GPL code again.

4865 The machine definition for the interpolation (G216 or G217) is missing

Cause
An attempt to move the axes with an ISO interpolation or the configuration indices have been set without defining in advance the configuration matrices and the axes that form the machine.

Solution:
Please correct and compile the GPL code again, using the instructions ISOG216.

4866 The index definition of the selected machine configuration (M6) is missing

Cause:
An attempt to move the axes with an ISO interpolation without defining in advance the indices of the machine's configuration matrices

Solution:
Please correct and compile the GPL code again, using the instruction ISOM6.

6.2.10 Errors generated by CNCTPA communication driver

16385 Disconnected module

Cause:
The connection between the Supervisor PC and a module was interrupted.
The possible causes are the following:
- power failure of the remote module
- Ethernet cable disconnection, even if temporary, due to a false contact in the connectors or to damaged cables
- power failure or malfunctioning of the Ethernet hub (if present)
- interruption of the remote module firmware due to damaged configuration files
- remote module CPU reset due to overheating or to EM disturbance

Solution:
Verify that the module is switched on. Verify Ethernet cables and connectors. Update the firmware in the remote module. Check that this module has not overheated because of insufficient ventilation and that it is not subject to EM disturbance. If the problem persists, please, contact the Manufacturer.

16386 Connected module

Cause:
A remote module was connected to the Supervisor PC after Albatros initialisation phase. Albatros tries to connect all the modules indicated in the System Configuration during booting, which lasts approximately 4 seconds. Any module connected later generates a system error.

16387 Reconnected module

Cause:
A remote module was reconnected to the Supervisor PC after being disconnected. This error always follows error 16385: "Disconnected module".

16388 Initialized module

Cause:
A remote module was reinitialized during normal functioning. This implies that the module was disconnected and reconnected to the Supervisor PC beforehand. Therefore this error always follows error 16385: "Disconnected module".
It indicates the module reset due, for example, to power failure.

16389 Module interrupted connection

Cause:
A remote module has closed the connection with Albatros. This may happen when the module does not receive any command or query from the Supervisor PC for a long time. This error shows a problem (overload or deadlock) on the Supervisor PC.

Solution:
Check the Supervisor PC for programs that may cause overloads or deadlocks. Disable the screen saver on the Supervisor PC. If the problem persists, contact the machine constructor.

16641 The control firmware does not respond to commands

Cause:
An error arose during system initialization. Specifically, firmware does not respond as expected. This fault might be caused by a damaged firmware file.

Solution:

Try to reset system and if necessary install Albatros again. If the problem persists, please, contact the Manufacturer.

16642 TpaSock does not respond to commands

Cause:
An error arose during system initialization. Specifically, communication software with remote modules does not respond as expected. This fault might be caused by a damaged software file.

Solution:
Try to reset system and if necessary install Albatros again. If the problem persists, please contact the Manufacturer.

16643 Operating System cannot use RTX

Cause:
The Operating System installed on the PC does not allow the use of RTX and consequently does not allow correct operation of versions of Albatros requiring its presence.

Solution:
Update the PC Operating System. Please, check the minimum system requirements on the installation manual of Albatros (InstallationGuide.pdf).

16645 Error sending firmware code

Cause:
An error arose during system initialization. Specifically, transmission of a firmware file to a module failed.

Solution:
Try to reset the system. If the problem persists, please contact the Manufacturer.

16646 Could not restart firmware code

Cause:
An error arose during system initialization. Specifically, firmware failed restarting after a previous stop.

Solution:
Try to reset system. If the problem persists, please contact the Manufacturer.

16897 RTX not installed

Cause:
The installed version of Albatros requires RTX installed on the PC; however, this has not been detected.

Solution:
Install RTX, or install it again if already loaded. Please, refer to the installation manual of RTX Albatros (InstallationRTXGuide.pdf).

16898 User has no Administrator rights

Cause:
Albatros was started up by a user without Administrator rights on the PC. Administrator rights are required for correct operation of Albatros.

Solution:
Please close current working session and access system as "Administrator" or as other user with Administrator rights.

16899 Wrong dimension of module RAM

Cause:
RAM dimension detected on remote module is inconsistent with expected dimension. This fault is normally caused by a hardware failure.

Solution:
If the problem persists, please contact the Manufacturer.

16900 Module IP address is wrong

Cause:
A remote module has been detected whose IP address does not belong to the supervisor PC subnet. Albatros cannot communicate correctly with the module.

Solution:
Check the settings of the AlbDHCP service and of the LAN board on the supervisor PC. Please read Albatros installation manual (InstallationGuide.pdf).

16901 Module is already connected to another plant

Cause:
A remote module appears to be connected to a different supervisor PC. This may be caused by presence on the network of another PC with Albatros running and using the same module. It may also be caused by a failure of the communication software on the module.

Solution:
Check that no other supervisor PC is using the remote module. Reset the module. If the problem persists, please contact the Manufacturer.

16902 The module is not configured

Cause:
A module appears not to be configured in Albatros "System Configuration".

Solution:
Please configure the module.

16903 Firewall settings prevent communication

Cause:
A firewall blocking communication between Albatros and remote modules has been detected.

Note: Albatros can identify Windows Xp firewall only and not other firewalls as those included in some anti-virus software packages.

Solution:
Please modify firewall settings or disable it.

16904 Network board not present or disabled

Cause:
No network board available for connection to remote modules has been found.

Note: the detection of a network board does not grant proper settings or connection.

Solution:
Check the network board and its configuration. If the problem persists, please contact the machine manufacturer.

16905 Control firmware code missing

Cause:
Albatros can't find a firmware file on the PC hard disk. The problem may be caused by an accidental file deletion or a incorrect update.

Solution:
Check that files in the FW sub-folder of Albatros setup are present and have the right version number. Please contact the machine Manufacturer.

16906 RTX version incompatible with control firmware code

Cause:
RTX version is not compatible with the installed firmware.

Solution:
Install the right RTX version or update the firmware. Please contact the machine Manufacturer.

16907 Operating system version is incompatible with control firmware code

Cause:
Operating system version of the remote moduleis not compatible with the firmware installed.

Solution:
Install in the remote module the correct operating system version or update the firmware. Please contact the machine Manufacturer.

17153 BoardType: Firmware code of GreenBUS transmitter missing

Cause:
A firmware file could not be found in FW folder. Normally this depends on a file accidental erasure or on an incomplete or damaged installation.

Solution:
Reinstall Albatros after executing a backup of the system. Please contact the machine manufacturer.

17154 BoardType: Part of firmware code of GreenBUS transmitter missing

Cause:
File containing firmware of GreenBus transmitter resides in FW folder but appears to be damaged or incomplete.

Solution:
Reinstall Albatros after executing a backup of the whole system. Please contact the machine manufacturer.

17155 BoardType: Error sending bootstrap code of GreenBUS transmitter

Cause:
An error occurred during system initialization. Specifically, sending a firmware file to a module failed.

Solution:
Try to reset the control. If the problem persists, please contact the Manufacturer.

17156 BoardType: Error sending Main code of GreenBUS transmitter

Cause:
An error occurred during system initialization. Specifically, a firmware file failed being sent to a module.

Solution:
Try to reset system. If the problem persists, please contact the Manufacturer.

17157 BoardType: Bootstrap code missing

Cause:
A firmware file could not be found in FW folder. Normally this depends on a file accidental erasure or on an incomplete or damaged installation.

Solution:
Install Albatros again after executing a backup of the system. Please contact the manufacturer.

17158 BoardType: Main code missing

Cause:
A firmware file could not be found in FW folder. Normally this depends on a file accidental erasure or on an incomplete or damaged installation.

Solution:
Reinstall Albatros after executing a backup of the system. Please contact the machine manufacturer.

17159 BoardType: Error sending bootstrap code

Cause:
An error occurred during system initialization. Specifically, sending a firmware file to a module failed.

Solution:
Try to reset the control. If the problem persists, please contact the Manufacturer.

17160 BoardType: Error sending Main code

Cause:
An error occurred during system initialization. Specifically, sending a firmware file to a module failed.

Solution:
Try to reset the control. If the problem persists, please contact the Manufacturer.

17409 Could not send auxiliary executable

Cause:
This error may occur while updating firmware on a remote module. It may be caused by a momentary network failure but also by a damaged firmware on the module. This error message may include an error code.

Solution:
Try to switch off and switch back on the remote module and repeat the update procedure. If the problem persists, please contact the Manufacturer.

17410 Could not run auxiliary executable

Cause:
An error occurred during system initialization. Specifically, an auxiliary program could not start execution. The error message also reports the auxiliary program name and possibly an error code.

Solution:
Try to reset system. If the problem persists, please contact the Manufacturer.

17667 DLLName: Could not run firmware code

Cause:
An error occurred during system initialization. Specifically, firmware code could not start. "DLLName" corresponds to the software component that caused the error.

Solution:
Try to reset system. If the problem persists, please contact the Manufacturer.

17668 DLLName: Could not get pointer to shared RAM

Cause:
An error occurred during system initialization. Specifically, the communication channel with firmware could not be opened. "DLLName" corresponds to the software component that caused the error.

Solution:
Try to reset system. If the problem persists, please contact the Manufacturer.

17921 Could not send NODETPA

Cause:
This error may occur while updating firmware on a remote module. It may be caused by a momentary network failure but also by a damaged firmware on the module. This error message may include an error code.

Solution:
Try switching off and on the remote module and repeating the update procedure. If the problem persists, please contact the Manufacturer.

17922 NODETPA did not restart

Cause:
This error may occur while updating firmware on a remote module. It may be caused by a momentary failure but also by a damaged firmware on the module. This error message may include an error code.

Solution:
Try to reset Clipper module and repeat the update procedure. If the problem persists, please contact the Manufacturer.

17923 NODETPA not running

Cause:
A remote module has been detected on the network, whose communication software is not running. Normally this is caused by a failure of communication software. The error message may include an error code.

Solution:
Try to switch off and on the remote the module. If the problem persists, please contact the Manufacturer.

18177 NODETPA tried to access an invalid address

Cause:
The remote module communication software raised an error. The error message may include an error code.

Solution:
Try to switch off and on the module. If the problem persists, please contact the Manufacturer.

6.3 Generic Notifications

6.3.1 Albatros starts running

It reports Albatros started and displays some useful information about the program version and the execution environment.

6.3.2 Albatros ends running

It reports Albatros is about to end running.

6.3.3 Computer enters stand-by mode

It reports that computer is about to entering stand-by mode. From now on Albatros is no longer able to respond to requests and notifications from GPL loop.

6.3.4 Computer exits stand-by mode

It reports that computer has just exited stand-by mode. Albatros resumes execution without being restarted.

6.3.5 Computer shutdown

It reports that computer is about to be shut down, while Albatros is still running.

6.3.6 Current access level

It reports that the access level to the Albatros functions has changed; it usually happens for maintenance operations or changing the loop or the configuration.

6.3.7 Software update of modules

It reports that it has been requested to update the software and firmware present in the remote controls.

6.3.8 Sending configuration to the modules

It reports that it has been requested to update the configuration and loop present in the remote controls.

7 System Configuration

7.1 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
- 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.

7.2 Device Configuration

7.2.1 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
Input port	512
Output port	512
Axis	240
Timer	128
Counter	128
Flag Bit	1024
Flag Switch	256
Flag Port	256

7.2.2 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
- Output Port
- Flag Port
- Timer
- Counters

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.
- **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 status of the device.
- **Public:** specifying if the status of the device can be read or modified by a GPL code not belonging to the group of the device.

7.2.3 Digital output

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

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.
- **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 status of the device.
- **Public:** specifying whether the status of the device can be read or modified by a GPL code not belonging to the group of the device.

7.2.4 Analog input

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

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.
- **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 Diagnostics windows or in the Synoptic Views.
- **Write accesses:** specifying the minimum access level required to modify the status of the device.
- **Public:** specifying if the status of the device can be read or modified by a GPL code not belonging to the group of the device.

7.2.5 Axis

Basic Data

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 boards 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), ***Stepper***, ***Digital***, ***Count*** (only encoder reading), ***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.
- **Phase Reverse**: it allows correcting via software a possible cable inversion of the encoder phases.
- **Reference Reverse**: it allows to reverse the speed reference of the axis. If used with the encoder phases reverse it allows 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:

- **Maximum speed**: maximum speed of the axis.
- **Acceleration**: time of the acceleration ramp.
- **Deceleration**: time of the 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 to reduce the loop error at equal speed.
- **Feed Forward Accel.**: percentage of feed forward acceleration. It allows 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.

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**: specifying the maximum configuration speed which can be used in manual movements. It will never exceed the maximum set speed.
- **Dynamic Servoerror**: enables or disables the dynamic servoerror. The default value is disabled dynamic servoerror, so the threshold servoerror remains enabled. See SETMAXERTYPE GPL instruction.
- **Reference speed** and **Loop error**: these two values are used to calculate the actual linear ratio of the two values in the machine. For the values to be considered, they both must be positive and different from zero, and the **Dynamic Servoerror** field must be enabled.
- **Wait while the axis stops**: 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 ENABLESTARTCONTROL GPL 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**: 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**: 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 status of the axis.
- **Public**: specifying whether the status of the axis can be read or modified by a GPL code not belonging to the group of the axis.

Axis chaining

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

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

Linearity correctors

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**: 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.
- **Data for Correction**: 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.

7.3 Logical Configuration

7.3.1 Plant Configuration

To define a new machine or modify an already existing one, you need to access the Module Configuration screen page. The Module Configuration is the configuration of the modules forming the plant.

The Configuration environment can only be opened from manufacturer level or higher.

Access to Configuration

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

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

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

To add a module to the plant simply press **[New]**. The button **[Modify]** allows modifying the data of an existing module, the button **[Delete]** allows removing a module, and the button **[Close]** allows quitting the plant configuration.

The data that identify the machine, and that are to be specified, are:
- the number of module: a sequential integer number that, if not specified, is assigned by the system
- a brief description

It also contains some data concerning the underlying Hardware, as follows:

- **Axis Control Frequency**: shows the rate at which the data are periodically exchanged between the numerical control and the devices connected to it through the field buses.
- **Number of Interpolation Channels**: shows the maximum number of interpolation channels (i.e. the max. number of the axis groups performing an interpolated movement) that can be managed simultaneously.
- **CPU use percentage**: shows the percentage of time, covering the period of axis control (i.e. the inverse of the "frequency axis control"), which is devoted to the execution of the firmware.

The same window can be opened from the group Configuration module branch, from the machine Configuration Module branch, and from the Hardware Configuration module branch.

7.3.2 Group Configuration

When the machine is designed from scratch, it is necessary to define all the components and to write all the control cycles. In many cases the design is carried out starting from a machine already manufactured, which is later changed according to the features of the new machine.

Creating a group

To create a new group access the Group Configuration screen page. All the groups, sub-groups and devices come from the first branch of the tree, which is the module. If you press the **[ENTER]** key or the **[Modify]** button, a dialog box opens to edit the module data.

Select the item ***Groups*** from the ***Edit*** menu

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

List of the commands to create, edit, delete, copy and paste groups, subgroups and devices

Command	Action
Create a new group, a subgroup, a device	[Ctrl+Enter], Button [New], Edit->New, context menu
Modify a group, a subgroup, a device	[Enter], Button [Modify], Edit->Modify…, context menu
Delete a group, a subgroup, a device	[Del], Button [Delete], Edit->Delete, context menu
Enable or disable the use of a group in the machine	Context menu, Button [Enable]
Copy a group, a subgroup, a device	[Ctrl+C], Button [Copy], Edit->Copy, context menu
Paste a group, a subgroup, a device	[Ctrl+V], Button [Paste], Edit->Paste, context menu

When you create a new group, the window appears and the following data must be set:
- The name of the group
- A comment (that can be translated into the languages managed by Albatros)

It is also possible to indicate the group as ***Intergroup***. At least one group must be set as intergroup, so this selection can be used by Albatros to identify the "main" group of the machine. This is the group, whose main function (that with the same group name) is automatically launched at start. This mechanism is used to start the machine and run the tasks, which verify that everything correctly works, before giving the control to the user.
When a group with devices connected to physical devices is disabled, you will be asked if you want to cancel the virtual-physical link. If you choose to keep the links, the pins of physical devices, which they are connected to, will be displayed in a grey colour on the graphic representation of the virtual-physical device.

Adding a subgroup to a group
To create a subgroup of the group, you must be positioned in the group.

If we do not intend to create any subgroups, select the *Device 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 subgroup. The process is similar to that used to create subgroups. In this case a window containing the list of available devices will appear.

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 Device Configuration.

Copying a device
The device copy function allows to make a copy of any device. First, select the device and then press **[Copy]**. To insert the device in the list, select the branch, where to paste the device and enable the command **[Paste]**. In the dialog window the device new name must be inserted.

Copying a subgroup
The function of copying a subgroup allows to copy a subgroup with all the devices it contains. To insert the subgrooup, select the branch, where to paste it and enable the command **[Paste]**. In the dialog window the subgroup new name must be inserted.

Copying a group

The function of copying a group allows to copy a group including all the subgroups and the devices it contains. Moreover, any group synoptic associated with it is copied (synoptic, whose name coincides with that of the group).
It allows to quickly create groups having a structure similar to that of an existing group, without having to recreate all the devices one by one. To copy a group, select the group you wish to copy, select the **[Copy]** button and enter the name of the new group in the dialog.

Copying of devices, subgroups and groups can also be done between different modules.

Choosing the groups belonging to the machine

After the creation of the group archive, the groups effectively present must be disabled or enabled.
The groups are all present in the machine, unless they are disabled through the button **[Disable]** or by selecting the same command from the context menu. If a group has been disabled, the option **Not present** appears next to the group name.

To display only the groups present in the machine, select the option ***Machine*** from the menu ***Edit***.
To insert a new group, press the button **[Insert]**. A window containing a group list present in the Groups archive and not yet inserted in the machine will appear.

At this point, select the preselected group and **drag it with the mouse** to the window of the Machine Configuration or select the button **[Insert]**.
Moreover, it is possible to remove an existing group through the button **[Remove]** or it is possible to start the research of the group name or of a device inside the machine composition structure.

In a machine, only one intergroup must be present.

7.4 Physical Configuration

7.4.1 System Configuration

The system configuration allows 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**.

- **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 Node 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.

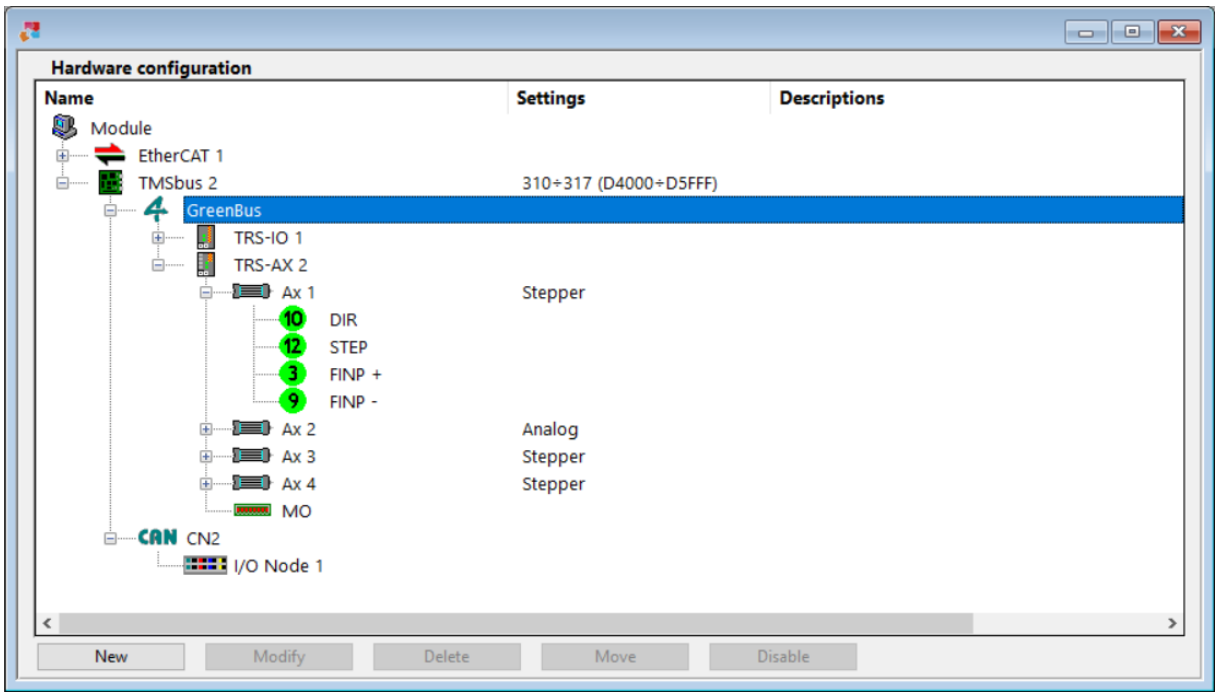
7.4.2 Hardware Configuration

In hardware configuration we define the boards and the nodes making up the system.
The board occupying the first position in the list is called Master board.

Types of configurable boards:
- TMSbus up to two
- TMSbus+ max. four
- TMSCombo+ max. four
- DualMech max. four
- DualMech Mono max. four
- TMSCan up to two
- TMSCan+ max. four
- AlbMech up to two
- EtherCAT one

Describing the hardware configuration window

The hardware configuration window opens if you select in the menu ***Edit->Hardware***.
To insert a board, a remote I/O module, or a CAN or EtherCAT node, press **[New]**. In this way a window appears where you can select the board or the remote I/O module, and, as far as it concerns the CAN or GreenBUS bus, the position where it should be inserted.
In general, no more than 4 boards per each module can be configured and, according to the type of board and bus, a variable number of remote I/O modules.



Hardware Configuration

The column **Settings** assigns some information concerning the board or the node.
With the **[Move]** command you can move a board or a node or an expansion of a TRS-IO or of a TRS-CAT from a position to a new one in the tree. The expansions can only be moved within the same node. This operation keeps the connections already present in the Virtual-Physical configuration.

Moreover, it is possible to disable a node or an expansion of a TRS-IO or of a TRS-CAT.
Disabling it causes the connections in the Virtual-Physical configuration to be maintained.
If the node belongs to a GreenBUS bus, the node and the devices connected to it are totally ignored by the system. Therefore, no error is generated if the module is not detected during initialization and no error is generated when a GPL instruction is executed on a device associated to the module.
If the node, or the expansion, belongs to the EtherCAT network, or if the expansion belongs to a GreenBUS bus, this must not be in the network. If we access an unconnected device in GPL, a system error will occur.
Consequently, this feature must be used with special care.

To disable a node or an expansion, use the **[Disable]** command, to enable a node or an expansion again, use the **[Enable]** command.

Default Configurations

Several default configurations are available. Either from menu **Edit->Modify control type** or the context menu on the Module branch it is possible to select the required configuration. In case the configuration is new, the tree is populated with defined boards and nodes; otherwise checks are performed to verify that the hardware already present is compatible with the selected module type. Everything that is not compatible is deleted.

Configure a node of a TPA bus

The I/O remote module types that can be configured on GreenBUS (v3.0) are as follows:

- Albre8 — 8 digital inputs and 8 digital outputs
- Albre16 — 16 channels configurable via software as digital input or output
- Albre24 — 24 digital inputs and 24 digital outputs
- Albre48 — 48 digital inputs and 48 digital outputs
- Albrem — 10 input ports and 10 output ports
- AlbSTEP — 8 digital inputs and 6 digital outputs, one stepping motor
- AlbEV — 20 or 24 solenoid valves (D-sub 25 pin connector)
- Albrea — 4 analog inputs and 4 analog outputs

The configurable types of remote module on GreenBUS (v4.0) are as follows:

- TRS-AX — 4 analog or stepper axes
- TRS-EV-24 — 24 solenoid valves (D-sub 25 pin connector)
- TRS-16 — 16 channels configurable via software as digital input or output
- TRS-IO — 16 channels configurable via software as digital input or output. This can be expanded through TRS-IO-E and TRS-AN-E modules (max. 5) and TRS-AC-E modules
- TRS-IO-E — 16 channels that can be configured via software as digital input or output; they can only be used as expansions of a TRS-IO module.
- TRS-AN-E — 1 analog input and 1 analog output that can be used only as expansion of a TRS-IO module
- TRS-REM — generic remote to produce special modules. Up to 12 input ports and 12 output ports can be connected.
- TRS-AC-E — 1 counting axis and 2 digital inputs, configurable as zero position reference and fast input. In the table below the maximum number of TRS-AC-E that can be configured in a TRS-IO is described.
  If there is a total of 3 expansions among TRS-IO-E and TRS-AN-E, one only TRS-AC-E expansion can be configured. If only one expansion (TRS-IO-E or TRS-AN-E) is configured, you can have up to 2 TRS-AC-E expansions.

The TRS-AX, TRS-IO, TRS-REM, 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.
The types of Tpa remote module that can be configured on EtherCAT bus 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, TRS-AN-E and TRS-AC-E modules.
- STAR-CAT — transforms an EtherCAT linear 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	Number of TRS-AC-E expansions
7	0
5	1
3	2
1	3

Regarding the TRS-AX remote modules, when we increase the number of inserted TRS-AX modules, the number of TRS-16 and TRS-IO that can be used decreases.
To calculate the maximum number of TRS-16 and TRS-IO remotes that 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 board, applying the formula we get: number of other remotes=32-(3*4), then a maximum of 20 remote modules of TRS-16 and/or TRS-IO type can be inserted.
The position of the remote should be chosen according to the address set through a switch on the remote module. Please, refer to the hardware documentation of the single remote.

If you select a TRS-AX remote, you may need to set the type of axes managed.

The following diagram describes which types of axes can be associated to the different hardware:

- AlbMech board digital axes
- DualMech board digital axes
- DualMech Mono board digital axes
- TRS-AX remote analog axes (if the specific connector type is Analog), counting axes (if the specific connector type is Analog), stepper axes (if the specific connector type is Stepper)
- AlbStep remote stepper axes
- TRS-AC-E expansion counting axes

In MECHATROLINK-II boards, each axis can be managed in position control or in speed control (default). The type of control for each axis can be modified in the Hardware configuration window in the **Settings** column. The number of configurable axes changes according to the axis control frequency value that has been set:

Board	**Axis Control Frequency (Hz)**	**Maximum number of servodrives**
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

Configure a node of a CAN bus

Bus control board

Albatros can manage devices on the CAN bus field through Tpa boards equipped with connector for CAN bus. The CAN bus is available on boards such as **TMSbus, TMSbus+, TMSCan+ and TMSCan**.

Configure the basic data and services
The CAN bus data is defined in the hardware configuration. Select the CAN bus, whose parameters must be defined and click the button **[Edit]**.
The basic data is:

- **Sampling time (TIME)**: sampling time in msec. It cannot be higher than 60000 (60 seconds). Default is 2. On S-CAN bus only 2 is accepted.
- **Time for synchronous communication of the PDO (TIMEPDO)**: time expressed in msec. It shows the time for the PDO synchronous communication. The value set cannot be higher than the TIME value (It is not a mandatory value).
- **Waiting time (TIMEAFTERRESET)**: time expressed in msec. It shows the waiting time during the initial phase due to a software reset of the nodes in the network. It cannot be higher than 60000 (60 seconds)
- **CAN cycle number without response (LIFETIMEFACTOR)**: this is the CAN cycle number without response to the Node Guarding call, before the error of disconnected node occurs. This value cannot be higher than 100 and lower than 1. (The default value is 3).
- **BaudRate (BAUDRATE)**: speed number of the CAN communication in kilobit per second (the value can be 1000, 500, 250, 125, 100).

The services can be enabled or disabled either by selecting or deselecting the option referring to the service. Some values with a meaning provided by the manufacturer of the machine can be applied to the **Extra** field. On this value, no control is performed. The default value is 16.

CAN node

Insert a new node

A node can be inserted by selecting the CAN branch of the tree and by clicking on the button **[New]**. The node type can be deducted from the bus type. If the bus is CAN, the node is an I/O Node, if the bus is S-CAN, the bus type inserted is Servo. In the dialog box, to insert the node, select:

- **Position**: this is the node number (from 1 on).
- **AutoOp**: if selected, the device acknowledges the automatic passage to the Operational status due to a reconnection.

Configure a node

For each node it is possible to define the PDO transmission and PDO reception. If the bus is CanOpen, up to 8 PDOs can be defined for the TMSBus and TMSBus+ boards, up to 4 for the boards TMSCan and TMSCan+. Select the node in the tree and click on the button **[New]**. The data to import in the dialog box is:

- **PDO type**: select **Transmission** or **Reception**, according to the definition of a TPDO or a RPDO.
- **Dimension**: dimension of the transmission or reception PDO.
- **COB-ID**: value that can only be defined on the boards TMSBus and TMSBus+. The value is displayed and stored in decimal. To display the value in hexadecimal notation, you need to select the **Hexadecimal** checkbox located next to the value itself.
- **Asynchronous**: if enabled, asynchronous PDOs are configured, that is, they are not updated at every cycle. This option is managed only on the boards TMSCan and TMSCan+. The reception of the asynchronous PDOs is made through the GPL code through the instruction RECEIVEPDO.

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 *cyclic synchronous speed mode* axis control.

The maximum number of EtherCat nodes is 200.

Introduction

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 sub-folder 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.

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".

EtherCAT hardware configuration

The hardware is configured by describing the master boards and, for each board, the list of physical devices connected to that board 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 must be inserted in the hardware configuration by selecting the Module branch and by clicking on the button **[New]**.

The EtherCAT node

To insert a node, select the board branch of the tree and click on the button **[New]**. The node index must be consecutive, so that at each command **[New]**, a node is added after the others. For each node, the following data must be defined:

- **Device name**: select the device name among those listed. Only the devices with the most recent revision are displayed; if you want to display all the revisions, you must select the entry **Display revisions**.
- **Axis mode**: select the operating mode to be used for the operating nodes, in compliance with what specified in the standard DS402, object 6060_{16} "Modes of Operation". The choice is between **Cyclic synchronous position** and **Cyclic synchronous velocity**. The latter is the default value.
- **Force as I/O node**: if enabled, it forces the numeric control to consider the drive as an I/O node. This attribute is only applied to nodes that support DS402 (servo-drives).

The axis mode is the only data that can be modified after inserting the node.

An EtherCAT node can be deleted, moved, disabled and copied. The commands are availbale in the context menu of each branch, in the main Albatros menu and in some buttons present in the window of hardware configuration.
The command **Delete** cancels both the selected node, all its expansions and virtual-physical links. All its following nodes are moved to the previous address.
The command **Move**, moves the selected node to a new address. The nodes following the chosen address are also moved by one position.
The command **Copy** stores the data of the selected node to use it by the command Paste later.
The command **Paste** inserts the node previously copied in the device tree. The node is inserted in the position selected by the user. All the following nodes are moved.

EtherCAT expansion
To insert an expansion select the node to which you want to add the expansion and click on the **[New]** button. The expansion indexes must be consecutive, so an expansion at the end of the others is added to each **[New]** command. Expansion PDO objects are not editable.

Description of a PDO

You can define up to 16 PDOs sent by the node (TxPDO) and up to 16 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 status word 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.

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

- Do not show any PDO.
  In this way the numeric control uses PDO configured by default in the device.

- Only show the PDO without providing any list of the objects.
  To be used when a CN has several PDO alternatives and not programmable.

- Describe the PDO in a complete way, by 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 node.

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.

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.

Modify a drive PDO

As for servo-drive nodes, there is a PDO for each axis, so that the nth TxPDO and the nth RxPDO of the node make reference to the nth axis of the node. The first two objects of each RxPDO and TxPDO have a preassigned significance and dimension. The following table shows how to configure the PDO of the axes of a same node, controlled in axis mode through **Cyclic synchronous velocity**.

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° axis	6040_{16}	$60FF_{16}$	6041_{16}	6064_{16}
2° axis	6840_{16}	$68FF_{16}$	6841_{16}	6864_{16}
nth axis	Add 800_{16} to each object of the previous axis.			

The following table describes how to configure the PDOs concerning the axes of the same node, controlled in axis mode through **Cyclic synchronous position**.

Drive	RxPDO		TxPDO	
	1° object 16 bit Controlword	2° object 32 bit Target position	1° object 16 bit Statusword	2° object 32 bit Actual position
1° axis	6040_{16}	$607A_{16}$	6041_{16}	6064_{16}
2° axis	6840_{16}	$687A_{16}$	6841_{16}	6864_{16}
nth axis	Add 800_{16} to each object of the previous axis.			

To modify the objects of an axis, select the axis in the tree and click on the button **[Modify]**. In the dialog box, the index of the transmission PDO, the index of the reception PDO and the list of the configured objects are displayed.
In particular, the data of the dialog box are:

- **Transmission PDO index**: this is the value of the transmission PDO of the axis. It is not a modifiable value.
- **Objects**: list of the objects of the transmission PDO. The first two objects are established according to the axis mode chosen.
- **Object list**: object list of the transmission PDO that can be used. For each object, if available, besides the index, also the sub-index and the description are defined.
- **Reception PDO index**: this is the value of the reception PDO of the axis. It is not a modifiable value.
- **Objects**: object list of the reception PDO. The first two objects are established according to the axis mode chosen.
- **Object list**: object list of the reception PDO that can be used. For each object, if available, besides the index, also the sub-index and the description are defined.

The buttons **[Up]** and **[Down]** are used to modify the order of the objects in the list of the window **Objects**.
The button **[Add]** allows to insert an object not available among those in the **Object list**.
To modify the value of an object, select the object and double-click with the mouse. The syntax to define a new object is:

- **object number**: this is the object number in hexadecimal form.

- **Subindex**: this is the sub-index number. It must be separated from the object number through the character '.' (point). This is an optional value that, if not defined, 0 is used as default value. This is in decimal form.
- **L**: object dimension in bits. It must be multiple of 8.

The values not available are obtained from the object dictionary data, or default values are assigned.

Example:
object also complete of sub-index and length: 1600.1L16
object without sub-index: 1601L24
simple object: 1603

Such values can be then read by the GPL through the GETAXIS instruction, to which reference should be made. It is possible, moreover, to trace the objects added both by the calibration window and by the oscilloscope.

More generally, from the GPL it is possible to access the writing and reading of objects in a PDO through the GETPDO and SETPDO instructions, to which reference should be made.

Additional PDOs

To define the reception and transmission additional PDOs, select the node in the tree and then click on the button **[Modify]**.

In the dialog box, to add a transmission PDO, click on the button **[Additional TPDO]**, while to add a reception PDO, click on the button **[Additional RPDO]**.

The dialog box to configure an additional PDO is similar to the dialog box to insert objects.

In this box, you can select the PDO index and the objects to be associated. If the PDO has mandatory objects, that are not modifiable, they are displayed and all the modification buttons are disabled. The new PDO is added in the tree after the drives. To modify the data, select and click on the button **[Modify]**. To delete an additional PDO, select the PDO in the tree and then, the button **[Delete]**.

Automatic acquisition of EtherCAT nodes

If the EtherCAT bus is present in the hardware configuration, it is possible to acquire the connected nodes from the network, using the command **Automatic acquisition of nodes** which can be activated from the context menu. In order for the command to be executed, the hardware configuration data and the data on the numerical control must be aligned and the ESI files that describe the node on the network must be present in the folder defined in Tpa.ini in the [tpa] section under the DirESIFiles item.

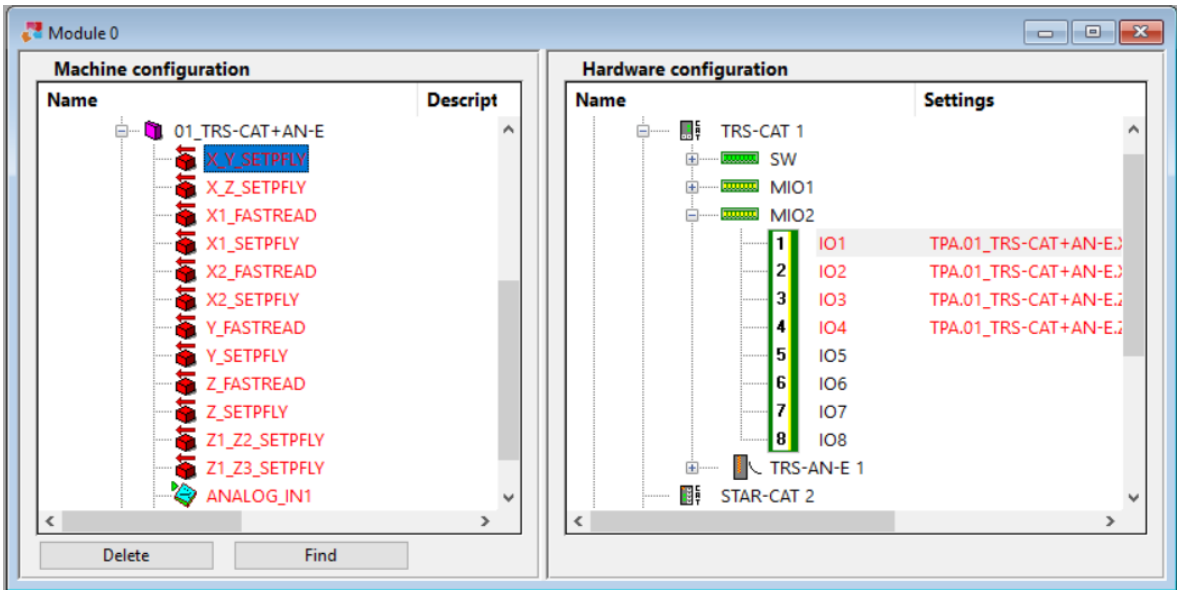
The acquisition of EtherCAT nodes from the network follows the listed rules:

- in hardware configuration no node is configured: the data related to the nodes are read from the network and displayed in configuration
- EtherCAT nodes have already been inserted in hardware configuration:
  - if the node read from the network has the same address, same Vendor ID, same Product ID and same Revision Number as the configured node, then this node is maintained with all its PDOs
  - if the node read from the network has the same address as the configured node, but different Vendor ID or different Product ID, then the configuration node is deleted, together with its virtual-physical and is replaced by the network node. If the network node is a drive, the PDOs are read for the ESI file, otherwise are read form the network node.
  - if the node read from the network has the same address, same Vendor ID, same Product ID and different Revision Number from the configured node, then only the revision number is replaced and the PDOs are maintained.
  - if the node defined in configuration does not have a corresponding node on the EtherCAT network that occupies the same address, then it is disabled in configuration and the virtual-physical is maintained.

7.4.3 Virtual-physical Configuration

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

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 links 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.

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 the EtherCAT bus is available in a module, it is still possible to configure boards for the MECHATROLINK-II bus, but with some limitations: with real-time at 1 ms, no more than six MECHATROLINK-II axes can be connected (for each bus); with real-time at 2 ms this limit increases to 16 axes.

The devices or the terminals still to be connected are marked in black. The devices marked in grey are those that belong to a disabled group, the configuration of which was maintained in the virtual-physical.

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 a suitable physical device** menu or the key combination **[Ctrl+space]**. 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 key combination **[Ctrl+NumPad+]** or the command **Edit->Find previous unlinked device** or the key combination **[Ctrl+NumPad-]**.

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 combination **[Ctrl+L]**.

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

7.4.4 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 links.

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 board or remote) was selected in the hardware window. When a component is selected, its name appears highlighted in blue.

7.5 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

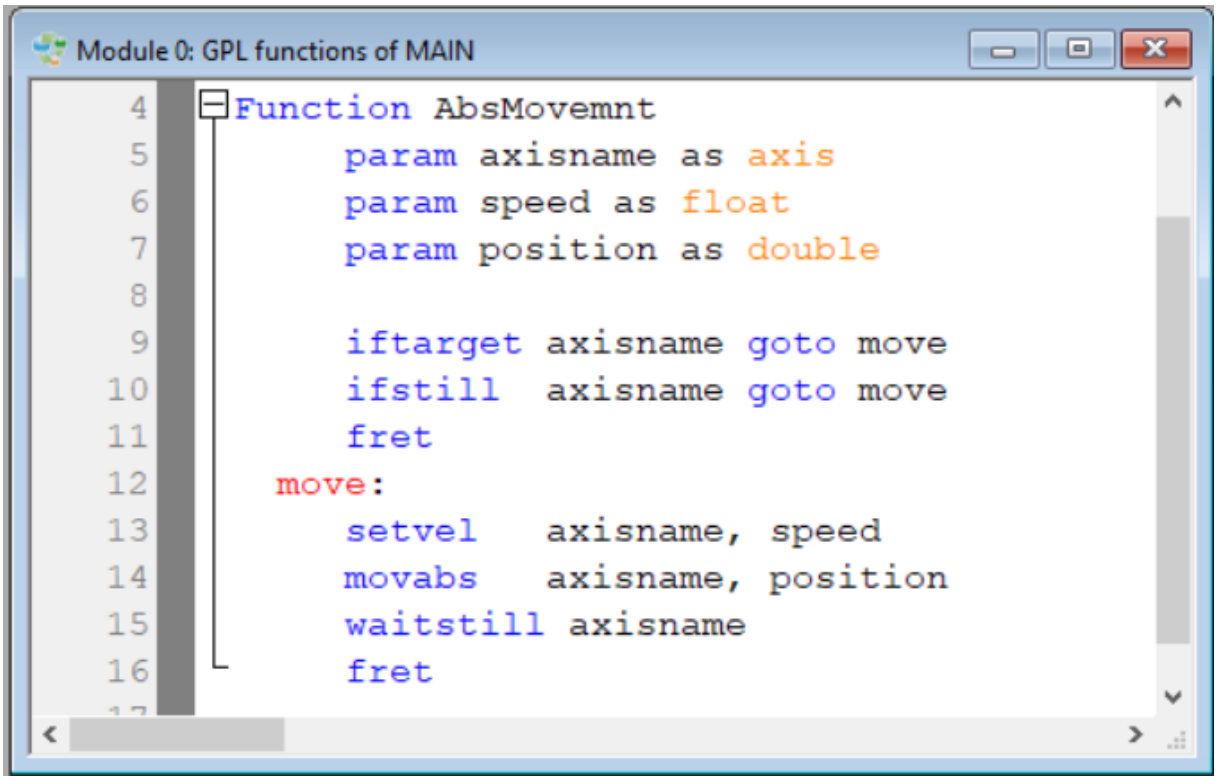
8 Development tools

8.1 Editor GPL

8.1.1 GPL Editor functions

GPL editor is the instrument that allows 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 status, before the last operation performed.
Redo	The situation is reverted to the older status 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, case-sensitive feature, research of a whole word, research through regular expressions.
Find next	allows the repetition of a previous search, enabling the change of the research criteria, set by with the command Find.
Replace	allows 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



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.
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 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]
```

Use of regular expressions

We can use the regular expressions in the **Find** and in the **Replace** windows. A regular expression is a sequence of numerical and alphanumerical characters used to find and replace portions of text in a larger text. Albatros uses the ECMAScript regular expression grammar. In this paragraph we will describe the main uses of the regular expressions in Albatros.

The simplest regular expression is the one consisting of a single character. Exceptions do exist, represented by the following characters:

- **. (dot)**: the dot finds any character. For instance "A.." finds any occurrence of a capital A followed by any two characters.

- **[] (square brackets)**: the square brackets allow specifying a list of characters within them. Occurrences of each character within will be searched for in the text. If the first character is **^ (caret)**, all characters will be searched for, except those listed in the brackets.

  For instance:
  [<>]: all occurrences of the characters < and > are searched for.
  [.]AX: all occurrences with the string "AX" within are searched for.
  [a-d]: all occurrences of characters a, b, c, and d are searched for. The hyphen indicates a set of characters.
  [\[\]]: all occurrences of character [and of character] are searched for.
  [^+]: all the characters are searched for, with the exception of character +.

- *** (asterisk)**: all occurrences of a character (or sets of characters) before the asterisk are searched for. The asterisk affects only the character before it: in order to have it affect a set of characters, we need to use parentheses.

  For instance:
  ;-*: all the occurrences of the characters **;** and **;-** and **;--------** are searched for.

- **+ (plus sign)**: it searches for one occurrence or more of the character before it. It differs from the asterisk, as the character before the + sign must always be present. The plus sign only affects the character before it: in order to have it affect a set of characters, we need to use parentheses. We will take the same example used for the asterisk:

  ;-+: all the occurrences of the characters **;**- and **;---** are searched for. The character **; (semicolon)** alone will not be searched for.

- **? (question mark)**: it makes the character before it optional, which, therefore, can be present once at most.
  For instance:
  Setfeedi?: all occurrences of the word Setfeed and of the word Setfeedi are searched for.

- **{} (curly brackets)**: they specify how many times a character (or a set of characters) needs be present in the text.
  For instance:
  ee{2}: all occurrences of two sequential ee are searched for.

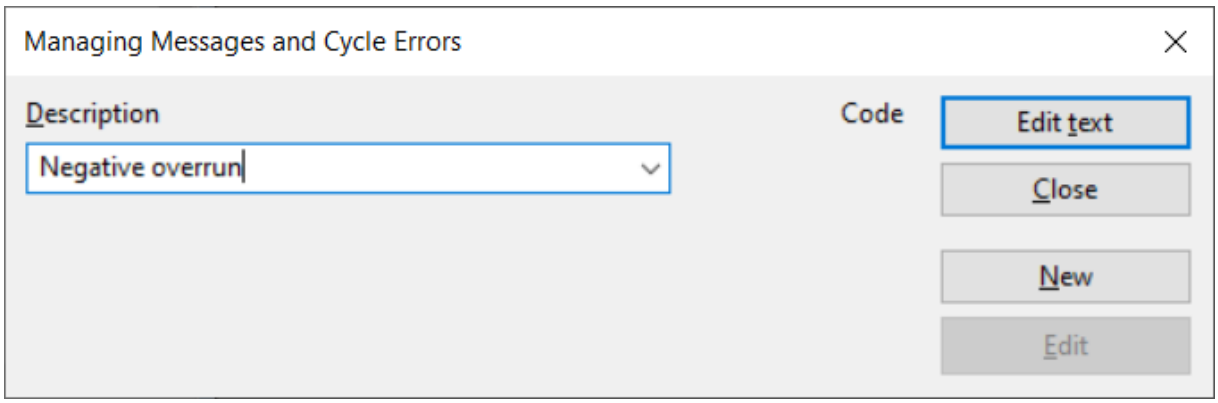
- **^. (caret and dot)**: it finds the first character of each line.

- **^ (caret)**: it finds the searched for term only if it is at the start of a line.
- **| (pipe)**: it finds the terms present both before and after the '|' character.
  For example:
  Send|Const: all the occurrences of the word Send and of the word Const will be searched.
- **\ (backslash)**: the backslash has two functions:
  1) it changes the normal character into a function.
     \b it finds the initial or final boundary of the word
     \B it finds the word, except its initial boundary
     \d it finds only the digits
     \D it finds all but the digits
     \s it finds the space
     \S it finds all but the space.
     Examples
     \bi: it finds all the words in the text starting with letter i.
  2) it changes a special character into a normal character. For instance, in order to search in a text for the special character **^ (caret)**, it will suffice entering the backslash before it.

8.1.2 Insert a 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.



Message management window

This way, there is no need to modify the file. The message will be inserted in the current language and, later, it will have to be translated into the other languages (using the program TpaLangs).
All the messages in the language file are listed under the heading **Description**. To insert a message inside the function, choose the required text and select the **[Edit text]** button.
To modify an existing message **[Edit]** or create a new one **[New]**, first type in the modification or the new text and then press the corresponding button.

8.1.3 Cryptography

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

Cryptography is enabled by setting Tele+=1 in tpa.ini. The default value is 0, which stands for encryption not enabled.

When a function file is saved and cryptography is enabled, the following message will be displayed: "Do you want to encrypt the file?". If you choose [Yes], the file will be encrypted. It is possible to encrypt a file that was previously saved unencrypted, while an encrypted file will always remain as such.

With encryption enabled, if the password is a daily Manufacturer type, an unencrypted file will not be encrypted.
An encrypted function file may only be displayed or edited in Albatros by the user who had previously saved it. The owner of an encrypted function file cannot change!

In case we wanted to decrypt a file, we would have to use an external program, called SBIANCA, located in the Bin folder of Albatros.

A way to use the program Sbianca is through the graphical interface. You select the files you want to decrypt. The **Status** and **Credentials** properties are displayed for each file. The **Status** may either be **Plain text**, if the file is unencrypted, or **Encrypted**, if the file is encrypted. The **Credentials** property gives information on the file visibility. **Freely readable** means that the file can be displayed with the current password level, otherwise it is **Blocked**, meaning that the file cannot be displayed.
Once the files are selected, you need to click on **[Decrypt!]** to decrypt them or **[Encrypt!]** to encrypt them.

A second way to use it is from command prompt. Commands and files are passed as arguments to the program. If there are no arguments, the graphical interface mode is activated. Its syntax is:
[-l|-e|-d] file1 file2 ...
With:

- -l, or no option, to display the status of each file
- -d to decrypt the files
- -e to encrypt the files

When the execution is complete, the program sends the results of the requested operation as output.
The program output is in markdown format.

8.1.4 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 Down arrow Right arrow Left arrow	moves the cursor to the selected direction
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+ Shift+Up Arrow Shift+Down Arrow Shift+Left Arrow Shift+Right Arrow	selects a rectangular code group

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 Ctrl+Ins	copy the selected text into the Clipboard
Ctrl+X Shift+Del	deletes the selected text and copy it into the Clipboard
Ctrl+V Shift+Ins	inserts the content of Clipboard from the cursor position

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/Undo

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

8.2 Libraries

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.

All the operations concerning the library are managed through a dialog window. 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 libraries removed by **[Erase]** are moved to the Recycle Bin of Window.

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.

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

8.3 Debug

8.3.1 The debugger

The debugger is a function of Albatros which allows 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
Stop Debugging	debug usage. The function file that was being debugged is opened in Edit mode.

To access the debug, 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 to modify the code. The background of the window is grey and the line in execution is highlighted in yellow.

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

8.3.2 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.

8.3.3 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. When we select a function, the file in which it is defined is opened and the cursor is positioned on the first instruction of the function. This allows to set Breakpoints even before starting execution.
You can debug any task and function without parameters.

Below we describe the meaning of the symbols used in the composition of the task execution tree. A particular 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

8.3.4 Show call stack

During debug it is possible to display the list of functions that were called but have not yet 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.

To observe the behaviour of a function call:
- move the cursor 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.

8.3.5 Breakpoints

A breakpoint allows 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.

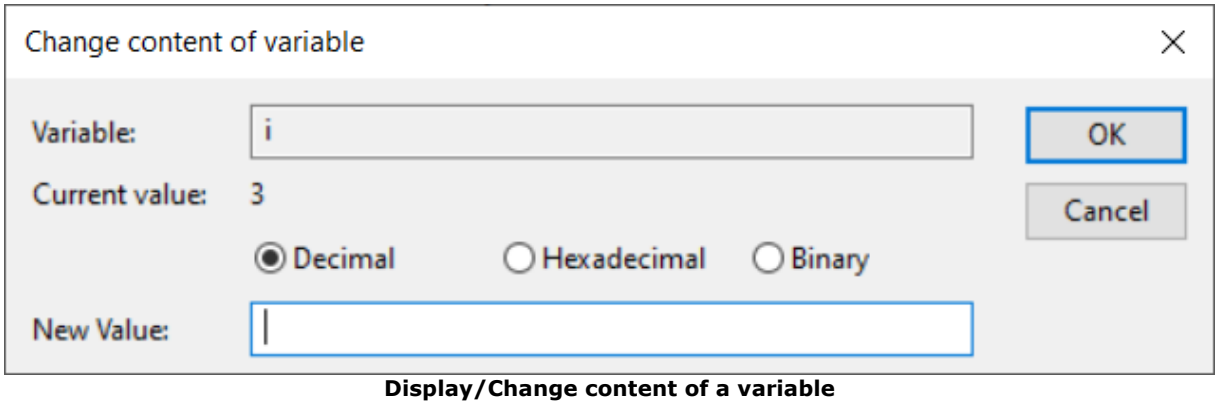
List of breakpoints

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.

8.3.6 Variable content

This command can be selected from the menu ***Debug->Variable Value***.
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 status of devices and device parameters



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 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 tool-tip 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 tool-tip; 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.

8.3.7 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

8.4 Control initialization

8.4.1 Network Connections

The profile machining of Albatros is protected by a USB hardware key, configured by TPA.
This command can be selected from the menu ***Cnc->Network Connections***. It displays the status 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 followed by the serial number of the remote module.

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).

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 System configuration for which a "Manufacturer" password level is required.
However, the module must be configured as "remote ALBRTX" in System configuration, beforehand.

Software update of a remote module
You can fully update the control software, available in the internal storage of the remote module, by selecting **[Update].**
Before applying this update, make sure that the selected remote module is connected to Albatros.

8.4.2 Hardware Diagnostics

This command can be selected from the menu ***Cnc->Hardware Diagnostics***. Hardware Diagnostics displays the list and the status of configured modules, of boards and of the nodes belonging to them, as defined in the hardware configuration. If the symbol of a board or of a node 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 board or node, but it does not match the type defined in configuration.

EtherCAT network topology

In the hardware diagnostics window, when in the *tree* a node of an EtherCAT network is selected, the button **[Details]** will be activated, which in turn activates the graphical view of the topology of the EtherCAT network.

In this graph, the info on the nodes that are physically present on the network is shown: the status of each node, the status of the axes, whether the node is a servodrive, and the quality of the communication. Each node and each axis is represented as a rectangle, whose colour defines its status.

Moving the cursor of the mouse on the rectangle, a tooltip will show describing the status and the communication errors of the node or the status of the axis.

Viewing and editing objects in the nodes

In the hardware diagnostics window, when in the *tree* a node of an EtherCAT network is selected, the button **[Object dictionary]** is activated, to view and edit the objects of the node. Editing data is only possible at **Manufacturer** level.

The objects are grouped according to their address:

Starting address	Final address	Area name

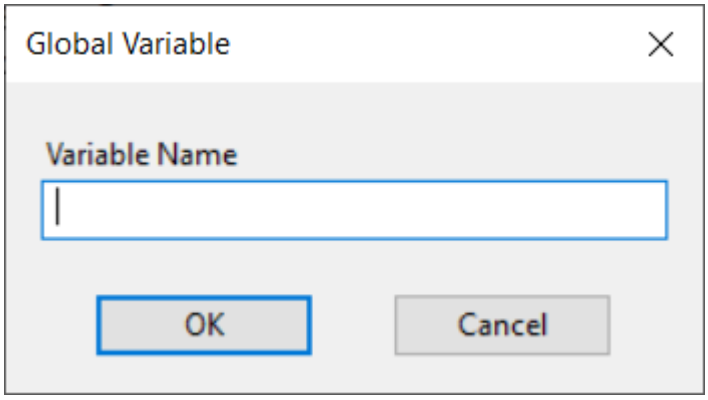
0x0000	0x0FFF	Data type area
0x1000	0x1FFF	Communication area
0x2000	0x5FFF	Manufacture specific area
0x6000	0x6FFF	Input area
0x7000	0x7FFF	Output area
0x8000	0x8FFF	Configuration area
0x9000	0x9FFF	Information area
0xA000	0xAFFF	Diagnosis area
0xB000	0xBFFF	Service transfer area
0xC000	0xEFFF	Reserved area
0xF000	0xFFFF	Device area

The integer values can be viewed both as decimals and as hexadecimals. When their value is changed, the number can also be inserted as hexadecimal, using the notation $…H, and in binary, with $…B (as it is done in GPL).

8.5 Test

8.5.1 Print global on disk

This command can be selected from the menu ***Test->Print global on disk***.
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

8.5.2 Start function

This command can be selected from the menu ***Test->Start single 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.

8.5.3 Message Import

Group messages assigned through the GPL DEFMSG instruction are saved in an .xmlng file. The messages inside the file can be edited, added, or canceled. In order to visualize the changes in the GPL code, you need to use the command ***Test->Import group messages,*** that is enabled only when there are no open windows.

To import group messages, all the GPL code must be compiled without mistakes. Otherwise, the user would be prompted with a message saying "GPL code has not been entirely compiled".
Group messages belonging to encrypted files cannot be imported (See chapter ***Development tools->Editor GPL->Cryptography***), therefore, the user is not allowed to visualise them in plain text.

Only the messages that were already 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.

All the languages listed below can be used in XMLNG files
"AFK" Afrikaans
"ARA" Arabic
"AZE" Azerbaijani
"BAS" Bashkir
"BEL" Belarusian
"BGR" Bulgarian
"BSB" Bosnian (Latin alphabet)
"BSC" Bosnian (Cyrillic alphabet)
"BRE" Breton
"CAT" Catalan
"CHS" Simplified Chinese
"CHT" Traditional Chinese
"COS" Corsican
"CSY" Czech
"CYM" Gaelic
"DAN" Danish
"DEA" German (Austria)
"DEU" German (Germany)
"ELL" Greek
"ENG" English
"ENU" English (United States)
"ESP" Spanish
"ETI" Estonian
"EUQ" Basque
"FAR" Persian
"FIN" Finnish
"FRA" French
"FPO" Filipino
"FRB" French (Belgium)
"FYN" Frisian
"GLC" Galician
"HAU" Hausa
"HEB" Hebrew
"HRB" Croatian (Bosnia-Herzegovina)
"HRV" Croatian (Croatia)
"HUN" Hungarian
"IBO" Igbo

"IND" Indonesian
"IRE" Irish
"ISL" Icelandic
"ITA" Italian
"JPN" Japanese
"KAL" Greenlandic
"KOR" Korean
"SAH" Sakha
"KYR" Kyrgyz
"LVI" Latvian
"LTH" Lithuanian,
"LBX" Luxembourghish
"MNN" Mongolian
"NON" Norwegian Nynorsk
"NOR" Norwegian Bokmål
"NLB" Dutch (Belgium)
"NLD" Dutch (Netherlands)
"OCI" Occitano
"PLK" Polish
"PTB" Portuguese (Brazil)
"PTG" Portuguese (Portugal)
"RMC" Romansh
"ROM" Romanian
"RUS" Russian
"SKY" Slovak
"SLV" Slovene
"SQI" Albanian
"SRM" Serbian (Latin alphabet, Serbia)
"SRN" Serbian (Cyrillic alphabet, Bosnia-Herzegovina)
"SRO" Serbian (Cyrillic alphabet, Serbia)
"SRP" Serbian (Latin alphabet, Montenegro)
"SRQ" Serbian (Cyrillic alphabet, Montenegro)
"SRS" Serbian (Latin alphabet, Bosnia-Herzegovina)
"SVE" Swedish
"TAJ" Tajik
"TRK" Turkish
"TTT" Tatar
"TUK" Turkmen
"UKR" Ukrainian
"URD" Urdu
"UZB" Uzbek
"VIT" Vietnamese
"WOL" Wolof
"XHO" Xhosa
"YOR" Yoruba
"ZUL" Zulu

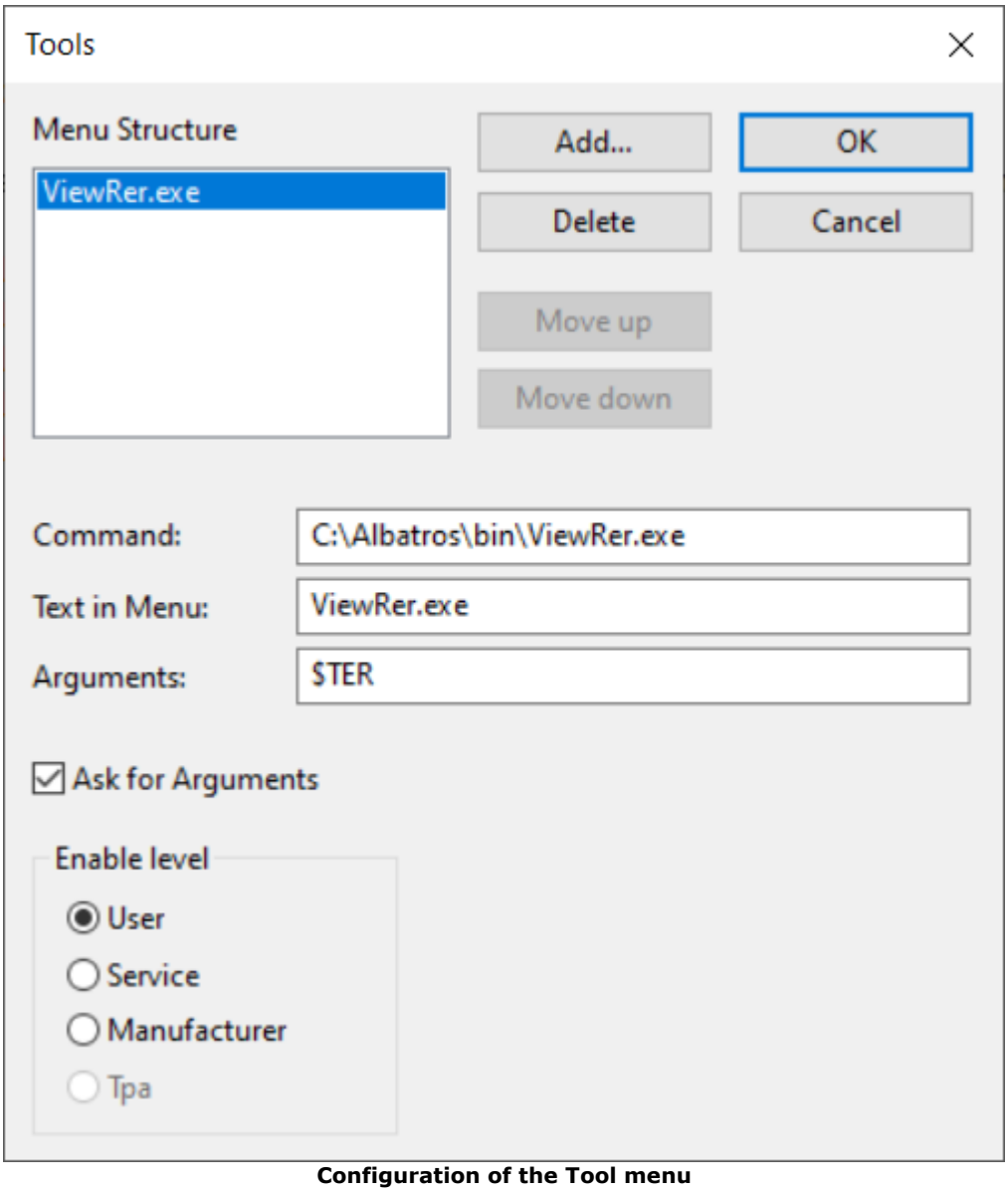
8.5.4 User notice in the alarm report file

With the command **Insert note**, from the menu **Test**, it is possible to insert a notice in the alarm report file of the current month (MONTHxx.TER). The menu option is enabled with a password level equal to or greater than service.

8.6 Tools

8.6.1 Customize...

This command can be selected from the menu ***Tools->Customize...***.
It allows to set a maximum of 20 programs whose execution can be started by Albatros ***Tools*** menu.



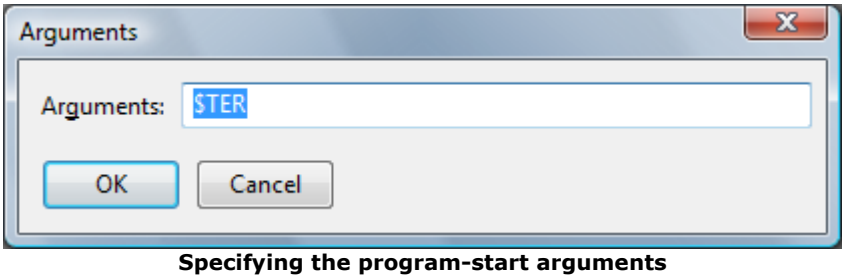
Configuration of the Tool menu

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 that should appear 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 example, by using the string $TER during ViewRER execution report file of current month open.
Here is the argument 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	CompletePath name of initialization file tpa.ini
$ReqDirModule	Path (disk and folders) of Albatros module. If several modules are configured, the module dialog box opens.

$ReqModule	Albatros module number If several modules are configured, the dialog box of the module number opens.
$ReqFile	Name of the file. A window requiring the name of the file opens. The selected file will be passed between the arguments of the program that will be executed.

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



Specifying the program-start arguments

Enable level: it sets the display level of the program in the ***Tools*** menu. Albatros 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.

8.7 Browser

8.7.1 The browser

Albatros 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*** or ***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.

8.7.2 Source browser

This command can be selected from the menu ***Debug->Source browser***. The identifier search opens a dialog window that allows 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.

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.

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

8.7.3 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)

9 Accessory programs

9.1 XConfMerge: program to merge the configuration file

XConfMerge is a tool that performs the merging of the configuration files. It is run from the command line of the bin folder, as XConfMerge reads the file tpa.ini.

The files read by XConfMerge are:
- hardware.xconf: it contains the data of the virtual-physical and of the hardware configuration (physical)
- devices.xconf: it contains the data of the configuration of groups, subgroups, and devices (logical)
- devices.xmlng: it contains the translatable messages. Anyhow, all language files present in the folder are considered
- addresses.xdb: it contains the logical addresses of the devices.

The arguments to be passed are:
- the folder from which to read the new files
- the number of module to which the custom refers. If no number is indicated, 0 is considered as default.

The path from which to read the file that is to be updated, that is the same on which to write the file gotten through the merging procedure, is inferred from the data set in tpa.ini.

Warning: the files to be updated are overwritten and no automatic backup is carried out.

Merging rules of the configuration files of groups:

1. if in both files there is the same group, subgroup, or device, the data and activation of the old file are kept.
2. if the group, subgroup, or device is only in the new file, the data and activations are copied.
3. if the group, subgroup, or device is only in the old file, this is cancelled.

Merging rules of the hardware and virtual-physical configuration file:

1. if in both files there is the same hardware, the hardware and the virtual-physical of the new file are kept with the activations defined in the old file.
2. if the hardware is only in the new file, the new hardware is kept with its activation and its new virtual-physical.
3. if the hardware is only in the old file, it is cancelled together with the virtual-physical.

Merging rules of the logical address files:

1. if the file is not in the folder of the files to be imported, then the custom logical addresses are kept and new addresses are assigned to the new devices, if there are any.
2. if the file is in the folder of the files to be imported, then the logical addresses contained in it are read and used.

Merging rules of the message files:

1. No merging is carried out, but the new message file is copied in the module folder.

At the end of the execution, the merging tool returns the following values:

0	all is OK
1	the file merging was unsuccessful
2	no arguments were defined

3	the module number entered as argument is wrong
4	the folder indicated as first argument does not exist

9.2 XParMerge: program to merge two parameter files

XParMerge is a tool that performs the merging of the parameter files.

It is run from command line on the bin folder, as XParMerge reads the file tpa.ini. The arguments to be passed are: the name of the new file and the number of the module it refers to.

The path from which to read the file to be updated and the one on which to write the file, gotten with the merging procedure, is inferred from the data set in tpa.ini.

Warning: no automatic backup of the old parameter file is performed, but it is overwritten.

Merging rule of Technological Parameter files:

1) if there is the same control in both files, the value of the old file is kept, but the other parameters defining it are updated (disabled, visible, GPL variable name.....) taking them from the new file:

2) if a control is defined in the new file that does not exist in the old file, the control is kept;

3) if a control is defined in the old file that does not exist in the new file, the control is cancelled.

Merging rules of Tool Parameter files:

1) if the reference dialog of the new file is different from the reference dialog of the old file, all the dialogs of the new file are kept, if there are and the ones of the old file are updated (new controls removed or added);

2) if the reference dialog of the new file is the same as the reference dialog of the old file, the dialogs of the new are added to the dialog of the old;

3) if the reference dialog is only present in the old file, this and its dialogs are cancelled.

10 GPL Language

10.1 Basic Features

10.1.1 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.

10.1.2 Conventions and terminology

Main adopted 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 *parametrised argument*.
CONSTANT	A fixed argument defined by means of the CONST meta-control or an argument which is rigidly fixed within the instruction.
VARIABLE	An argument defined as module 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 subprogramname".
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.
status	logic status, 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 variable 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

target position	Current "target" 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 target 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 target position to reach exactly the final value.
arrival threshold	Programmable interval whose central point corresponds to the final target position: when the real position enters this area, the movement is considered concluded.
arrival high threshold	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 target 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 betweenn the current speed and the 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 reducing, at equal speed and equal proportional gain, the value of the loop error.
feed rate override	Percentage of programmed speed. This parameter allows reducing 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 blocks of displacement.
backlash	Space between the cogs of a couple of gears.

10.1.3 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 instructions which allow 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 bytes. In the second case a maximum string size is defined.
The string values are sequences of characters delimited by double quotation marks. Example: "Press the button".
To enter the '"' character (double quotation mark), enter it twice. Example: Press the ""Start""". button.
To enter characters using the numerical code, write inside the string the \u characters followed by the numerical value of the hexadecimal character. Example: \u20ac is the symbol of Euro. If you write "\u20ac 15,6", you get € 15,6.

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 menu option ***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 technological parameter 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.
Examples 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 Kbytes.
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 module global variables can be defined as "nonvolatile".
The maximum size of variables memorized on nonvolatile RAM is 65536 bytes. The maximum size of a single non volatile matrix is 1024 bytes.

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 limit observance in the 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 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 default values are:
- READ reading for service (S or SERVICE)
- WRITE writing for manufacturer (M or MANUFACTURER) and TPA (T or TPA)

10.1.4 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 instructions 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
```

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

Predefined constants with preassigned value

The GPL language has some predefined constants. These can be used directly, they do not need to be defined. 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**

Predefined constants with preassigned value upon Albatros start

The GPL language includes some predefined constants whose value is defined when Albatros is launched. They can be used in the instruction IFDEF.

_ID_MODULE	current module number. The module number is between 0 and 15.
_REMOTE_MODULE	module type. The constant is 1 if the module is remote, it is 0 if the module is local, it is not defined if the module is not configured in the System configuration.
_VER_MAJOR	Albatros major version number. If the Albatros version is 3.2.1, the major version value is 3.
_VER_MINOR	Albatros minor version number. If the Albatros version is 3.2.1, the minor version value is 2.
_VER_REVISION	Albatros revision number. If the Albatros version is 3.2.1, the revision value is 1.
_VER_SP	string that describes the Service Pack, if installed; for example, "Service Pack 1f", otherwise it is not defined.
_VER_FULL	full version number. In case of version 3.2.1, it is $00030201H.

10.1.5 Keywords

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

The 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.
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= or READ	An attribute of functions or variables. It indicates the read access level. See Functions, Variables and Access rights
W= or WRITE	An attribute of functions or variables. It indicates the write access level. See Functions, Variables and Access rights

CONST	It allows 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

10.1.6 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. The maximum number of declarable functions is 8191.

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. It shares some properties with the devices: a univocal name (untranslatable), 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
Access rights allow 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 default values 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, the function 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 status of a digital input: instruction ONINPUT
- Change of status 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 cannot 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, for instance, 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.

10.1.7 Device parameters

Device type parameters are special variables which allow 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
Function	FUNCTION
Generic device	DEVICE
Task	TASK

Example of axis parameter declaration and use:

```
Function test
  Param axis as axis

  MovAbs        axis,100
  WaitStill     axis

Fret
```

10.1.8 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 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 status). 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 status of the axes.

A task is identified by the name of the GPL function 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 status:

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 status of the axes (axis 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 real-time 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 typical status 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.

10.1.9 Communications

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.

Typically it is used when the performances supplied by the Albatros OLE interface are not adequate.

On the GPL side IPC communication is implemented using the instructions SENDIPC, WAITIPC and TESTIPC.
In case of the local module the external processes may refer to the APIs supplied by RTX or to the COM component **gplipc2.dll** provided by TPA which makes its use easier.

In the case of a remote module, the processes running in the supervisor use the COM component

gplipc2net.dll.
For further information, please contact TPA.

10.1.10 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*
timeout	*double*
analog input/output	*float*
director cosines	*double*
string control character	*char*
acceleration/deceleration	*integer*

10.1.11 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 categorize them according to their technological characteristics.
The mechanical components are:

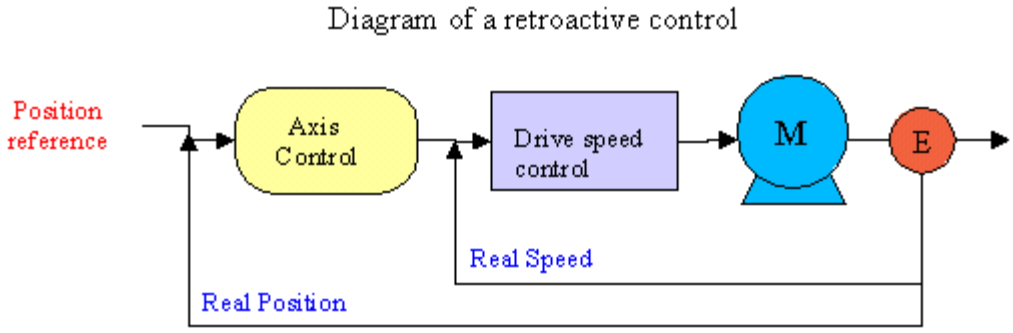
- frame
- guides
- bearings
- screws + ball screws

whose function is to counterbalance the forces involved, reduce friction, turn rotational motion into translation motion, etc.

The electric and electronic components are:

- motor
- limit switches
- encoder
- tachometric dynamo

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

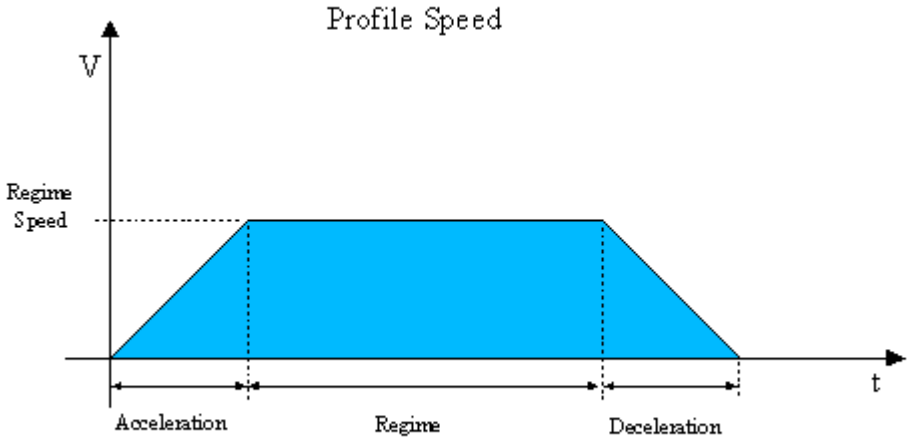


The function of the numerical control is to control the position and the movement of the axes.

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 the 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 the 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 "quiescent threshold" (that determines tolerance for axis positioning). If this is not done within 5 seconds of the expected end of movement, the system generates a "not ended movement" 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 dividing the speed profile in time intervals equivalent to the axis refreshment time and calculating the area of each part. The area corresponds to the position increase the axis has to reach in that frame 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 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***.

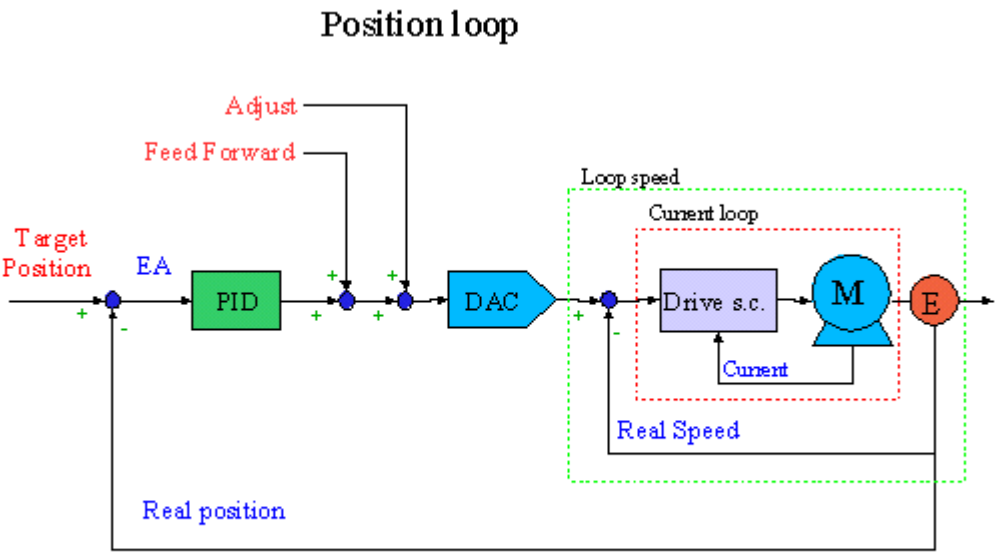


Diagram of Albatros axis control

10.1.12 Linearity correctors

The table of an axis linearity correctors is seen as a matrix, whose name is *GroupName.SubgroupName.AxisName#*correctors, or *GroupName.AxisName#*correctors, and it can be used in all the instructions that have access to matrices, matrix elements, and matrix rows.

The number of matrix columns corresponds to the number of axes that were set in the linearity corrector window, in the axis configuration. The automatic correctors are entered in the first column. All the correction values are float. The number of total matrix rows can be obtained through the GPL instruction LASTELEM.

These matrices can be accessed both in reading and in writing and, if modified, their values are immediately used to correct the position, only if the correction is enabled.

Example:

```
Function ReadCorr
local i as integer
local j as integer
local row as integer
local column as integer
local firstvalue as float

; reading of the first automatic correction value of axis AX
firstvalue = X.Ax#correctors[1][1]
; axis number in the linearity corrector window
setval 3 column
lastelem X.AX#correctors row
; increase all corrections of a constant value
for i 1 row
     for j 1 column
       X.Ax#correctors[i][j] = X.Ax#correctors[i][j] + 0.025
     next
next

fret
```

10.1.13 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 using TpaLangs, which is a program external to Albatros that manages the message files. This program supports the translation of the messages in the different languages.

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 to insert the text of a message directly from Albatros, without having to use TpaLangs.
A second option for message handling in various languages is using the GPL DEFMSG instruction.

10.1.14 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 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.

10.2 Special functions

10.2.1 Axis movement customization

Albatros system graphical interface allows 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. However, it may be necessary to customize the functions, for instance, to define axis movement restrictions, depending on the machine status, or to manage auxiliary devices, like 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 should be in the same sub-group which belongs to the reference 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 must belong to the same subgroup of the referred 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#AxisName#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.

10.2.2 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 kinds 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
```

10.2.3 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 "OnUIEnd#" function is 2 seconds, then Albatros will terminate all the tasks.

The maximum downtime this function has to finish the execution can be configured in TPA.ini., under [Albatros] section, under Timeout.OnUIEnd=*value*, where *value* is in milliseconds and cannot be greater than 60000.

10.2.4 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.

10.2.5 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. What follows is read during this time:

- Cycle errors
- System errors
- Messages.

At the end of the execution, Albatros closes.

The maximum downtime this function has to finish the execution can be configured in TPA.ini., under [Albatros] section, under Timeout.OnUIUnplugged=*value*, where *value* is in milliseconds and cannot be greater than 60000.

10.3 Instructions

10.3.1 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 its functioning
- eventual notes
- eventual examples

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

10.3.2 Types of instructions in the GPL language

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

Instructions to manage Input/Output

GETFEED	reads the override feed rate
INPANALOG	reads an analog input
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 status of a flag bit or flag switch
MULTIWAITINPUT	waits for the status of various inputs
OUTANALOG	modifies an analog output
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 status of a flag bit or flag switch
WAITINPUT	waits for the status of an input
WAITPERSISTINPUT	waits for a persistent status of an input

Instructions to manage the Axes

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
SETQUOTE	sets the position
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
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 status 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 to manage the Axis Parameters

Writing/Reading

DEVICEID	writes the logical address associated to a device
GETAXIS	reads one or more data of an axis

Point-to-point movement

SETACC	sets acceleration
SETDEC	sets the deceleration
SETDERIV	sets the coefficient of derived action
SETFEED	sets the point-to-point feed rate
SETFEEDF	sets the feed forward
SETFEEDFA	sets the acceleration feed forward
SETINTEG	sets the coefficient of integral action
SETMULTIFEED	sets the percentage value of feed rate override of the affected axes
SETPROP	sets the coefficient of proportional action
SETSLOPE	sets the type of ramp of the rapid movements
SETVEL	sets the speed

Interpolated movement

LOOKAHEAD	sets the interpolation lookahead
SETACCI	sets the acceleration for interpolation
SETACCLIMIT	enables and disables the automatic calculation of the interpolation steady status speed
SETACCSTRATEGY	selects the type of acceleration
SETAXPARTYPE	changes the axis parameter set currently in use
SETCONTORNATURE	sets the contouring angle
SETDECI	sets the deceleration for the interpolation
SETDERIVI	sets the coefficient of interpolation derived action
SETFEEDFAI	sets acceleration feed forward in interpolation
SETFEEDI	sets feed forward in interpolation
SETFEEDFI	sets the feed forward in interpolation
SETINTEGI	sets coefficient of interpolation integral action
SETPROPI	sets coefficient of interpolation proportional action
SETSLOPE	sets the type of ramp in the movements in the interpolation
SETSLOWPARAM	changes the parameters to calculate the slowdown speed in the event that the slowdown functionality while contouring is active
SETVELI	sets the interpolation speed
SETVELILIMIT	sets the individual components of the specified axis speed

Coordinated movement

SETFEEDCOORD	sets the percentage value of the highest instantaneous variation of the axis feed rate.
SETOFFSET	enables a position offset

Chained movement

RATIO	sets the chaining ratio of a slave axis with respect to its own master
SETDYNRATIO	changes dynamically the chaining ratio during the movement of the master axis.

Generic parameters

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
NOTCHFILTER	sets the notch filter cut-off frequency for the specified axis
RESLIMNEG	disables the negative limit of the axis
RESLIMPOS	disables the positive limit of the axis
SETADJUST	enables axis adjust
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.
SETDEADBAND	sets the minimum voltage for the affected axis
SETENCLIMIT	changes the incorrect encoder connection limit
SETINDEXEN	enables or disables on the axis the reset of the position that corresponds to the zero position reference
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)
SETMAXERTYPE	sets the type of the test on the servoerror
SETPHASESINV	activates or deactivates the phase inversion on axis
SETREFINV	enables or disables on the indicated axis the the inversion of the speed reference
SETRESOLUTION	changes the resolution if an axis

Instructions to manage the Counters

DECOUNTER	decrements a counter
INCOUNTER	increments a counter
SETCOUNTER	sets a counter

Instructions to manage the Timers

HOLDTIMER	locks a timer
SETTIMER	sets a timer
STARTTIMER	starts the timer

Instructions to manage the Matrices

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 to manage the Strings

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 to manage the Communications

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 the 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
SENDIPC	sends an IPC information
WAITIPC	waits for an IPC information
WAITRECEIVE	external data reception with standby

Mathematical instructions

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
XOR	binary XOR

Instructions for Multitask management

ENDMAIL	reports the end of the execution of a task
ENDREALTIMETASK	terminates a real-time task
ENDTASK	terminates a task
GETPRIORITYLEVEL	reads the priority level of the current task
GETREALTIME	returns time lapsed since the beginning of axis real-time
GETREALTIMECOUNT	returns the number of real-time lapsed
HOLDTASK	interrupts the execution of a task
RESUMETASK	resumes the execution of a task
SENDMAIL	sends a command to the 'mail' mailbox
SETPRIORITYLEVEL	sets the priority level of the current task
STARTREALTIMETASK	starts a real-time task
STARTTASK	starts the execution of a task

STOPTASK	interrupts the execution of a task and stops 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 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 status 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

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
TYPEOF	type of the argument
WATCHDOG	enables, updates, disables the watchdog from GPL on the TMSWD hardware module

Instructions to manage the MECHATROLINK-II

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, ALARM and IO_MON

Instruction to manage the standard field buses

AXCONTROL	sets a value for ControlWord
AXSTATUS	returns the value in the StatusWord
CNBYDEVICE	returns the node and board number of a device
READDICTIONARY	reads the content of a dictionary object
WRITEDICTIONARY	writes the content of the dictionary object

Instructions to manage the EtherCAT

ACTIVATEMODE	sets an operating mode
ECATGETREGISTER	returns the content of an ESC register (EtherCAT Slave Controller)
ECATSETREGISTER	writes the content of an ESC register (EtherCAT Slave Controller)
GETPDO	returns an object inside a PDO Ethercat
SETEOE	activates or deactivates the Sniffer
SETPDO	sets an object inside a PDO Ethercat

Instructions to manage the CAN Bus

GETCNSTATE	returns the status of the NMT protocol for a node of a CANOpenboard.
GETSDOERROR	returns the last error occurred
GETMNSTATE	returns the status of the NMT protocol for the master node of the CANoPen board.
RECEIVEPDO	reads the content of an asynchronous PDO
SENDPDO	writes the content of an asynchronous PDO
SETMNSTATE	sets the status of the NMT protocol for the node of the CANopen board.

Instructions for the Simulation

DISABLE	disables one or more axes
DISABLEFORCEDINPUT	disables the inputs to be forced
ENABLE	enables one or more axes
ENABLEFORCEDINPUT	enables the inputs to be forced
RESETFORCEDINPUT	forces an input to OFF
SETFORCEDANALOG	forces an analog input
SETFORCEDINPUT	forces an input to ON
SETFORCEDPORT	forces an input port

Instruction for the “Blackbox” functionalities

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

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 ruling the fluidity of the ISO interpolated movement
KINEMATICEXPR	sets the single expressions of inverse and direct kinematics

10.3.3 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 board, 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.
For all TPA boards controlling the feed rate a dedicated connector is available.

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**.

INPFLAGPORT

Syntax

INPFLAGPORT	**flagportname, variable**

Arguments

flagportname	name of flag port device
variable	variable

Description

It copies the status 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 status 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

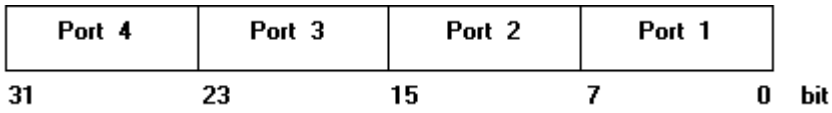
MULTIINPPORT **port1[,...,port4],variable**

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 up to 4 input 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.



MULTIOUTPORT

Syntax

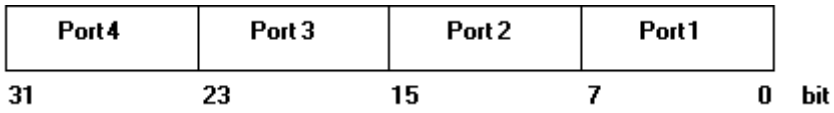
MULTIOUTPORT **value, portname1[,...,portname4]**

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.



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

Arguments

mask	constant or variable. Mask of involved flags
flag1[,...flag32]	name of flag device
status	predefined constant. Acceptable values are: - **ON** flag status: enabled - **OFF** flag status: 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 status indicated by the **status** 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 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], status [, timeout [, GOTO label]]**
MULTIWAITINPUT	**mask, input1[,..., input32], status [, timeout [, CALL subprogramname]]**
MULTIWAITINPUT	**mask, input1[,..., input32], status [, timeout [, functionname]]**

Arguments

mask	constant or variable. Mask of involved inputs
flag1[,...flag32]	name of input
status	predefined constant. Acceptable values are: - **ON** flag status: enabled - **OFF** flag status: 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 status indicated by the **status** 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 status. 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**.

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

Arguments

flagname name of flag device
status predefined constant. Acceptable values are:
- **ON** flag status: enabled
- **OFF** flag status: 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 status indicated by the parameter **status** (ON/OFF).
If the only optional argument present is **timeout**, the cycle error "**flagname** flag awaiting **status**" 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, status [, timeout [, GOTO label]]**
WAITINPUT	**nameinput, status [, timeout [, CALL subprogramname]]**
WAITINPUT	**nameinput, status [, timeout [, functionname]]**

Arguments

nameinput	name of input
status	default constant. Acceptable values are: - **ON** input status: enabled - **OFF** input status: disabled
timeout	constant or variable. Maximum waiting time
label	jump label (GOTO)
subprogramme	subprogram label (CALL)
functionname	name of function

Description
It waits for the **nameinput** input to be in the status indicated by the parameter **status** (ON/OFF).
If no optional arguments are specified, the cycle error "**Nameinput** digital input waiting **status**" 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, status, timepersist [, timeout [, GOTO label]]**
WAITPERSISTINPUT	**nameinput, status, timepersist [, timeout [, CALL subprogramname]]**
WAITPERSISTINPUT	**nameinput, status, timepersist [, timeout [, functionname]]**

Arguments

nameinput	name of digital input device
status	default constant. Acceptable values are: - **ON** input status: enabled - **OFF** input status: disabled
timepersist	constant or variable
timeout	constant or variable. Maximum waiting time
label	jump label (GOTO)
subprogramme	subprogram label (CALL)
functionname	name of function

Description
It waits for the **nameinput** input to reach the status indicated by the parameter **status** (ON/OFF) and to remain in that status for the time specified in **timepersist** (unit of measure: seconds).
If no optional arguments are specified, the cycle error "**Nameinput** digital input waiting **status**" 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).

10.3.4 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 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.
On its part, 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 stepper axes, 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 management of axis AxisName" is generated.
See also RATIO.

Example

```
; Y axis is chained to X axis
CHAIN          X, Y
; X axis moves.
; Y replicates the movement of X
MOVINC         X, 100
```

CIRCABS

Syntax

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

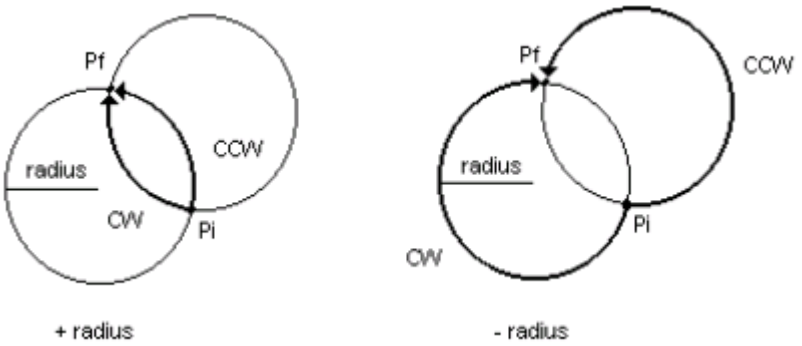
Arguments

label	constant or variable integer. Label identifying a displacement bloc
axis1, axis2	name of acis devices
position1, 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** counterclockwise
radius	constant or variable. It indicates the value of the radius of the circle
angle	constant or variable. It indicates the abgle 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 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 block.
Stepper 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 by the word interpolation we intend a coordinated movement of more axes affected by discrete error due to axis piloting method.



CIRCINC

Syntax

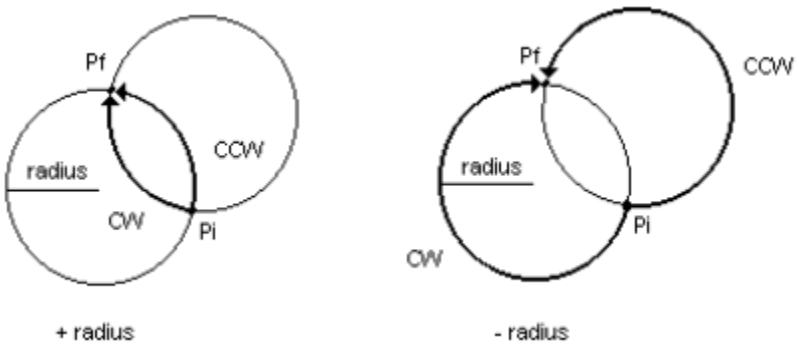
CIRCINC **[label],axis1, position1, axis2, position2, direction, ±radius [, angle]**

Arguments

label	constant or variable integer. Label identifying a displacement bloc
axis1, axis2	name of axis devices
position1, 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** counterclockwise
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 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 identify univocally the displacement block.
Stepper 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 by the word interpolation we mean a coordinated movement of more axes affected by discrete error due to axis piloting method.



CIRCLE

Syntax

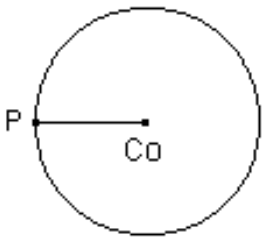
CIRCLE **[label],axis1, axis2, direction, radius, angle**

Arguments

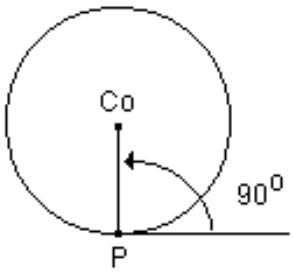
label	constant or variable integer. Label identifying a displacement bloc
axis1, axis2	name of axis devices
direction	integer variable. It specifies the kind of rotation. Acceptable values are: **CW** clockwise **CCW** counterclockwise
radius	constant or variable. It indicates the value of the radius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

Description

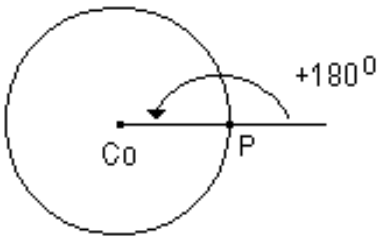
Complete circular interpolation.
It generates a circle with **axis1** and **axis2**, in the indicated direction, with the indicated **radius** and according to the set starting **angle**.
The **radius** can only have positive values.
The **angle** must be given according to the trigonometric convention, 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.
Stepper 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.



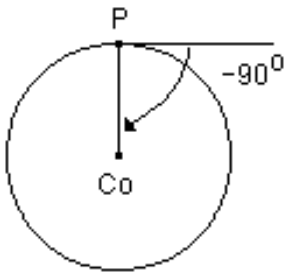
angle = 0



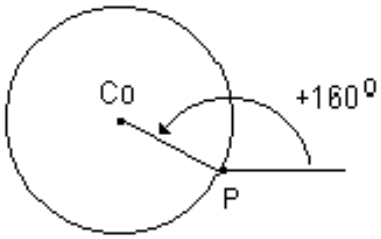
angle = 90°



angle = +180°



angle = -90° (o+ 270°)



angle = +160°

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 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 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 instruction.
The parameter **direction** allows 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** = number of rows that must be executed, **fin** = number of row preceding row that must be executed, **direction** = UP. Otherwise a system error is generated.

Stepper 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 management of axis" 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.

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
```

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

The position 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, takes axis to position 0.0
ENDMOV    X, 0.0
```

FASTREAD

Syntax

FASTREAD **axis1, status, variable1 [,axis2, variable2],[..., axis8, variable8]**

Arguments

axis1...[...axis8]	name of axis devices. Axis1 is the master axis
status	default constant. It can have 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 status.

If the indicated axes are analog, they must be part of the same board (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.
With an EtherCAT bus, for each axis swtiching encoder (please see the instruction SWITCHENC), the maximum number of allowed axes decreases by one. Once the limit is exceeded, the system error "4400 Too many active axes in FASTREAD" will occur. Moreover, the additional encoder must be connected to a TRS-AC-E expansion on Trs-CAT. Otherwise, the system error "4375 FASTREAD executed on axes from different boards" will occur.
The instruction ends when the input switches to the indicated **status (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 digital axes on board with MECHATROLINK-II bus stands on **EXTI2** input and does not 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) condition, disabling the *position control*. If the **axis** is slave in a sequence with other axis, the link is broken and the movement of the **axes** interrupted.
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/counterclockwise (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.
Stepper 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/counterclockwise (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 identify univocally the displacement bloc.
Stepper 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, status**

Arguments

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

Description

According to whether the parameter **status** 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.
Stepper 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.
Stepper 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 movement* 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 status even though physically it is not moving.
A move can be aborted with the instruction ENDMOV. In this case it cannot be restarted.

Note

1) 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 status "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 TPA.

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 status even though physically it is not moving.
A move can be aborted with the instruction ENDMOV. In this case it cannot be restarted.

Note

1) 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 status '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 TPA.

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 target position of displacement bloc end

Description

Absolute multi-linear interpolation up to 16 axes. This interpolation movement enables to advance the speed profiles, setting their respective tolerances on the axes by means of the instruction SETTOLERANCE (axis 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.
Stepper 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 block
setveli         x, velx1
setveli         y, vely1
setveli         z, velz1
multiabs        x, positionx1, y,positiony1, z,positionz1
; second block
settolerance    x,tolx2, y,toly2, z,tolz2
setveli         x, velx2
setveli         y, vely2
setveli         z, velz2
multiabs        x,positionx2, y,positiony2, z,positionz2
; third block
settolerance    x,tolx3, y,toly3, z,tolz3
setveli         x, velx3
setveli         y, vely3
setveli         z, velz3
multiabs        x,positionx3, y,positiony3, z,positionz3
; fourth block
settolerance    x,tolx4, y,toly4, z,tolz4
setveli         x, velx4
setveli         y, vely4
setveli         z, velz4
multiabs        x,positionx4, y,positiony4, z,positionz4
waitstill       x, y, z
```

MULTIINC

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 target 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.
Stepper 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 chaining.
When the system is switched on, all the configured axes set in free status and switch to normal status 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**).
This instruction is normally used after setting a rotational translation frame of reference 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 axis device

value	global variable of type integer

Description

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

Note

The variable **value** must be a group global variable or a module global variable.

SETPFLY

Syntax

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

Arguments

axis	name of axis device
status	predefined constant. It indicates the status 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 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 **status**. 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. The allowed values are: 0 = only the master axis zeroes the coordinate. The slave axis keeps the previous coordinate. different from 0 = master and slave synchronously zero the coordinate.

Description

This instruction enables to set, how the indicated slave axes will behave for a master setpfly instruction. The instruction 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 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

```
FREE            X
SETPZERO        X, 100
```

SETPZEROCHAINSTRAT

Syntax

SETPZEROCHAINSTRAT **axis, [value]**

Arguments

axis	name of axis device
value	Integer variable. The allowed values are: 0 = only the master axis zeroes the coordinate, the slave axis keeps the previous coordinate. different from 0 = master and slave 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 target 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

SETQUOTECHAINSTRAT

Syntax

SETQUOTECHAINSTRAT **axis, [value]**

Arguments

axis	name of axis device
value	integer variable. The allowed values are: 0 = only the master axis zeroes the new coordinate, the slave axis keeps the previous coordinate. different from 0 = master and slave synchronously zero the coordinate.

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.

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

position1_ax1...position3_ax3 director cosine of the three axes _ax3
axis1...axis3 name of devices. Axis type

Description

It allows activating an X' Y' Z' Cartesian reference system with a rotational translation with reference to the X Y Z absolute frame of reference 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 are 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

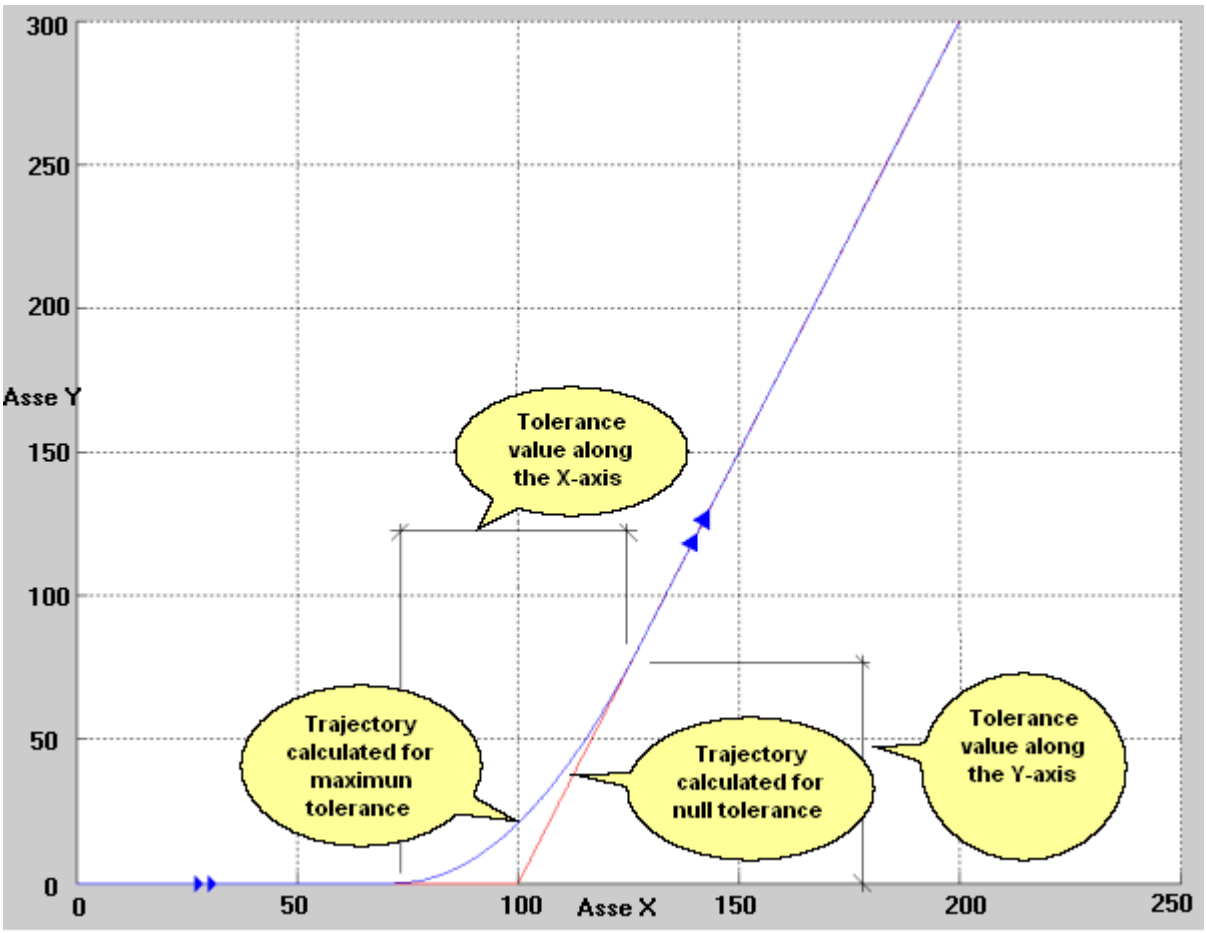
SETTOLERANCE **axis1, value1, [axis2, value2 [, axis3, value3 [,..., axis16, value 16]]]**

Arguments

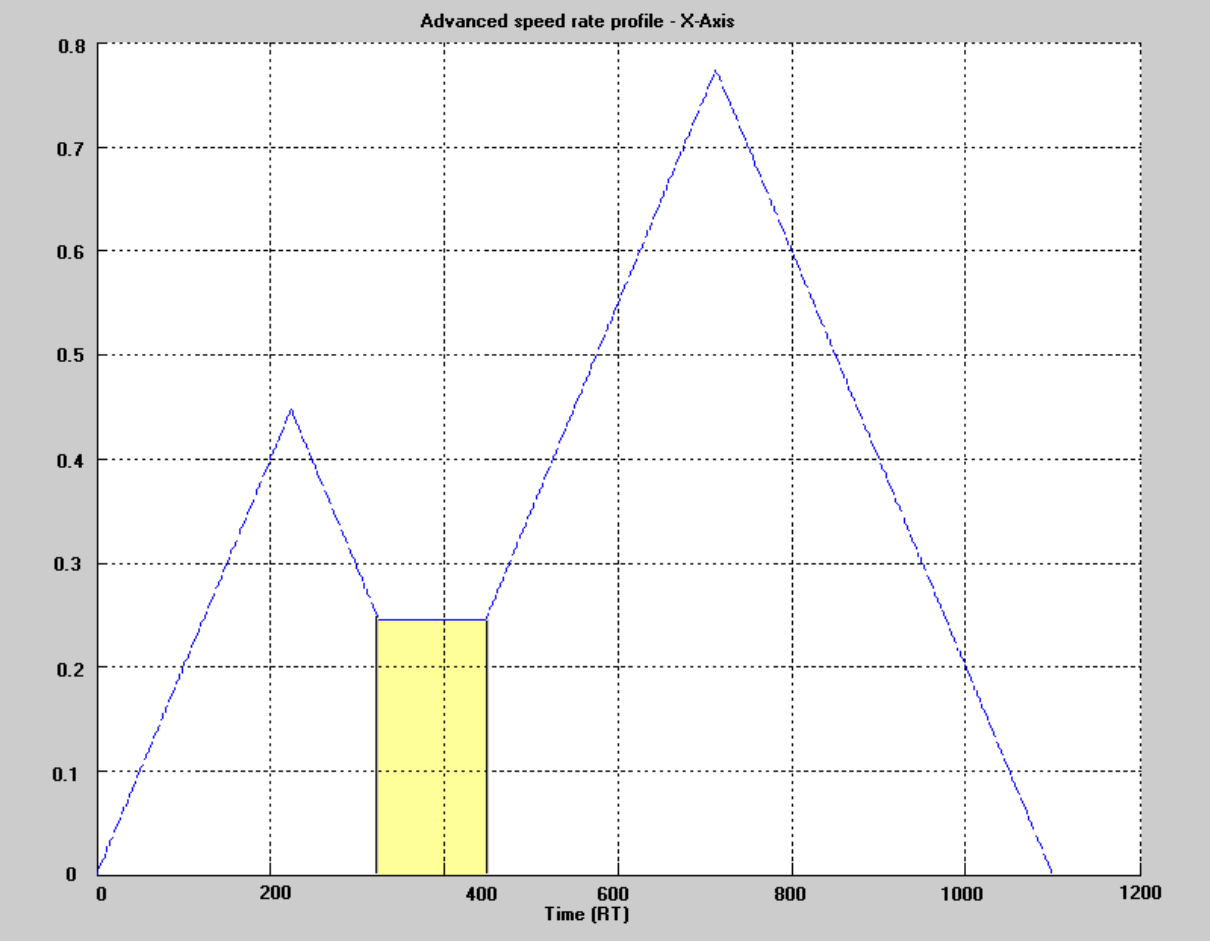
axis1...axis16 name of axis devices
value1...[...value16] constant or variable. Maximum tolerance value that can be applied to the axis.

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 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 the **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

It allows to replace the encoder of **axis1** with the encoder of **axis2**. The encoder is switched 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 switch is immediately executed, regardless of the axis position.

If only **axis1** is declared, the functioning with a single encoder is restored.

Axis1 cannot be a stepper, counting, and virtual type, **Axis2** can be a counting axis only. Furthermore, neither **axis1** nor **axis2** can be involved in movements in chain as slave axes.

The instruction generates system error 4101 – *Inconsistent management of axis AxisName*, when either **axis1** or **axis2** is declared as slave in a movement in chain, or when **axis1** is running a FASTREAD or SETPFLY instruction. Moreover, system error 4105 – *Instruction not executable on axis AxisName* may occur when the declared axis type is not among the possible ones.

WAITACC

Syntax

WAITACC **axis [,..., axis6]**

Arguments

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

Description

It waits for the acceleration status or one of the subsequent statuses of all the indicated **axes** (1÷6).
The task where the instruction is executed is put on waiting status, until the axis reaches the acceleration status or one of the subsequent statuses.

Axis statuses 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

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 "not ended movement". 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 stepper 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

Waiting for the deceleration status or one of the subsequent statuses on all the specified **axes** (1÷6).
The task where the instruction is executed is put in stand-by, until the axis reaches the deceleration, coordinate, wait on the higher range, wait on the lower range and wait while the axis stops statuses.

Axis statuses are identified by an integer:
- acceleration = 1
- steady = 2
- deceleration = 3
- coordinate = 4
- wait on the higher range = 5
- wait while the axis stops = 6
- wait on the lower range = 7

WAITREG

Syntax

WAITREG **axis1 [,..., axis6]**

Arguments

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

Description

It waits for the regime status or one of the subsequent statuses of all the specified **axes** (1÷6).
The task where the instruction is run is put on hold until the axis is in the statuses of steady, deceleration, coordinate, wait on the higher range, wait on the lower range and wait while the axis stops statuses.

Axis statuses are identified by an integer:
- acceleration = 1
- steady = 2
- deceleration = 3
- coordinate = 4
- wait on the higher threshold = 5
- wait while the axis stops = 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 status).

Example

Axis Homing routine

WAITTARGET

Syntax

WAITTARGET **axis1 [,..., axis6]**

Arguments

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

Description

It waits for the current target position of all the specified **axes** (1÷6) to reach the target position. The real quote will not match the target 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 status or one of the subsequent statuses of all the specified **axes** (1÷6).
The task where the instruction is executed is put on wait status, until the axis reaches the wait on the higher range, wait on the lower range and wait while the axis stops statuses.

Axis statuses are identified by an integer:
- acceleration = 1
- steady = 2
- deceleration = 3
- coordinate = 4
- wait on the higher range = 5
- wait while the axis stops = 6
- wait on the lower range = 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.

constant	description	type	return value
_CFGTYPE	Axis typology	i	1=analog, 3=stepping motor, 4=digital, 5=count, 6=not used, 7=virtual
_CFGUM	unit of measure	i	0=millimetres,1=inches,2=degrees,3=rev.
_CFGRIS	resolution	d	impulses per _UM
_CFGVMAX	maximum speed	f	m/1' or inch/1" or degrees/1" or rev/1'

constant	description	type	return value
_CFGVMAXD	maximum speed in manual mode	f	m/1’ or inch/1” or degrees/1” or rev/1’
_CFGVMAXI	maximum interpolation speed	f	m/1’ or inch/1” or degrees/1” or rev/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	stepper ramp start speed	f	m/1’ or inch/1” or degrees/1” or rev/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
_CFGDSE	enable dynamic servoerror	b	0=disabled, 1=enabled

constant	description	type	return value
_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 while the axis stops	b	0=disabled, 1=enabled
_CFGENCTYPE	type of encoder	i	0=simulated or absent, 1=real
_SRPP	stepper ramp start speed	f	m/1' or inch/1" or degrees/1" or rev/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 rev/1'
_VELI	interpolation speed	f	m/1' or inch/1" or degrees/1" or rev/1'
_MODE	axis functioning mode	b	1=normal, 2=free, 8=interpol., 10=coord.
_PHINV	encoder phase inversion	b	0=no inversion, 1=inversion

constant	description	type	return value
_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	
_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 status	b	1=accel., 2=regime, 3=decel., 4=position, 5=wait high thres., 6=wait while the axis stops, 7=wait low thres., 8=start
_QT	Target 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

constant	description	type	return value
			the DAC axis board. 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 board. 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	
_GONETIME	elapsed time from the beginning of the movement	f	sec (0 for slave axis and stepper 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 stepper axis)
_TARGETTIME	time taken to generate the Target Position	f	microseconds
_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, stepper axes)
_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 stepper axis and ISO movement)
_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	type of motion in 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	

constant	description	type	return value
_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, 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 the displacement blocs still to be performed	i	
_EXECBLOCK	number of performed displacement blocks	i	
_TOTALBLOCK	total number of queued displacement blocs in the movement (current value)	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	
_LENSETPZERO	distance covered to reach the zero pulse	d	position
_TORQUEINST	instantaneous value of torque	i	
_CURRSLOPE	Returns the type of ramp currently used in the rapid movements	i	0=linear, 1='S' shaped, 2=double 'S' shaped
_CURRSLOPEI	Returns the type of ramp currently used in the interpolated movements	i	0=linear, 1='S' shaped, 2=double 'S' shaped
_CURRSLOPEI	Returns the type or ramp used in the interpolated movements	i	0=linear, 1='S' shaped, 2=double 'S' shaped

constant	description	type	return value
_AXISPAR1...AXISPAR8	Optional additional parameters for EtherCAT axis	i	number
_ISOMOVETYPE	Type of ISO motion in execution	i	0=quick ISO motion, 1=interpolated ISO motion, -1=other
_QMAINENC	Real Position of the main encoder when the secondary encoder is in use	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 stepping 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 stepping 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.

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 stepping 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.

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 stepping 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 **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 stepping motors.
See also instruction SETPROPI.

SETSLOPE

Syntax

SETSLOPE **axis [, value]**

Arguments

axis name of axis device
value constant or variable integer. Type of ramp.

Description

It sets the type of ramp to be used for the rapid movement:

- 0 linear ramp
- 1 "S" shaped ramp
- 2 double "S" shaped ramp

If **value** is omitted, the configuration ramp is restored.
The type of ramp can only be changed with stationary axis in POSITION. Otherwise the system error no. "4101 – Inconsistent management of axis AxisName" occurs.
You can check the type of the ramp currently set for the axis in association with this instruction, through the instruction GETAXIS and the parameter _CURRSLOPE.

See also the instruction SETSLOPEI.

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.
If 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.

Note:
By interpolation block we mean 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 it is omitted, the instruction 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] integer constant or variable

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.
The allowed values for the **value** parameter are 0, 1, and 2. If value 0 is passed, the usual acceleration strategy is adopted (the least of the axes involved in the interpolation is chosen as profile acceleration). In case of value equal to 2 and linear interpolation, the highest acceleration the individual axes can withstand is taken (considering the individual components of all the axes, either linear and/or rotary); the acceleration management in case of circular interpolation remains unchanged. The case of value 1 calls for an obsolete management that is kept for compatibility.

SETAXPARTYPE

Syntax

SETAXPARTYPE **axis, [value]**

Arguments

axis name of the axis type device
[value] variable or integer constant

Description

When a multilinear interpolation is performed, this instruction allows 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 parameters used are those of interpolation.
The parameter set change can only be made if the axis is still in POSITION status, otherwise the instruction generates the system error no. 4101 "Inconsistent management of axis AxisName".

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. So, 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*. The deceleration feature is disabled when the *maximum deceleration angle* is less or equal than the *maximum contouring angle*. The *maximum deceleration angle* is equal to 180 degrees. If a greater value is set, the generates the following error no. 4399: "Parameter out of range".
The deceleration feature is enabled only if the instruction JERKSMOOTH is active; however, the contouring is always active.

Nota

The use of this instruction is correlated to the use of the instructions JERKSMOOTH and SETSLOWPARAM, and 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 stepping 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 stepping 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.

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 stepping motors.
The **value** variable admits values included between 0 and 100.

See also instructions SETFEEDF, SETFEEDFA, SETFEEDFAI.

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 stepping 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 stepping motors.
See also instruction SETPROP.

SETSLOPEI

Syntax

SETSLOPEI **axis [, value]]**

Arguments

axis	name of the axis device
value	constant or variable integer. Type of ramp.

Description

It sets the type of ramp to be used for the movement in the interpolation (where allowed):

- 0 linear ramp
- 1 "S" shaped ramp
- 2 double "S" shaped ramp

If **value** is omitted, the configuration ramp is restored.

The type of ramp can only be changed if the axis has not been used in an interpolation channel, yet. Otherwise the system error no. "4101 – Inconsistent management of axis AxisName" occurs.
You can check the type of the ramp currently set for the axis in association with this instruction, through the instruction GETAXIS and the parameter _CURRSLOPEI; by means of the parameter _REALSLOPEI you can see the type of ramp used by the axis (the ramp of the channel in which the axis is engaged).

See also the instruction SETSLOPE.

SETSLOWPARAM

Syntax

SETSLOWPARAM **axis [,value1,value2]**

Arguments

axis	name of device of axis type
value1	double constant or variable. General reduction factor
value2	double constant or variable. 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.
Stepper 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	double constant or variable. It represents the maximum percentage of feed rate override.
value2	integer constant or variable. It represents the number of real-time, where the feed rate 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 **value2** variable. In other words, after applying a variation of feedrate override of **value1**, as highest value, by **value2** real-times, 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 **value1**=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**.

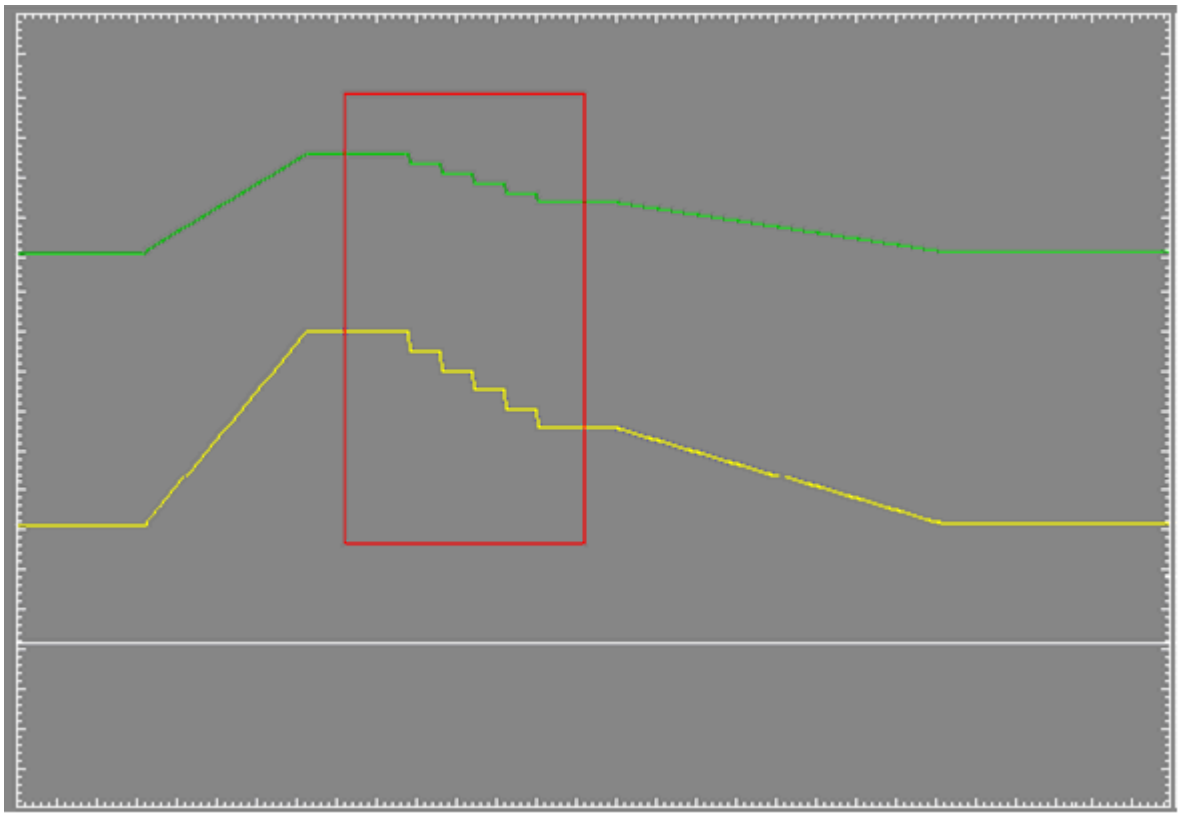
Example

```
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, rowInit, rowEnd,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 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

Syntax

RATIO **axis, [value]**

Arguments

axis name of axis device
value costant or variable. Reduction ratio.

Description

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 status and the corresponding master axis is not in position status.
See CHAIN instruction.

Example

```
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 type 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 status 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 management of axis AxisName", in the event that the **axis** declared is not a slave axis.

Generic Parameters

DYNLIMIT

Syntax

DYNLIMIT **axis, status**

Arguments

axis name of axis device
status predefined constant. Allowed values:
ON enabling dynamic controls of the axis limits
OFF disabling dynamic controls of the axis limits

Description

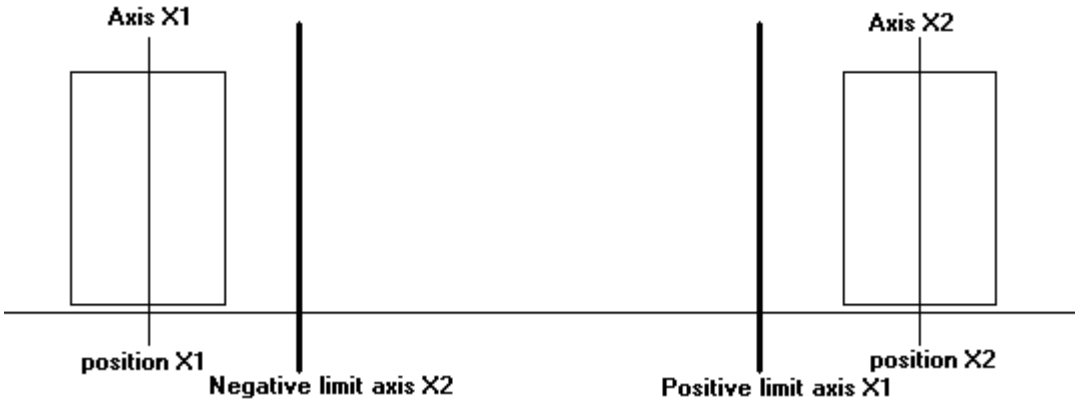
It enables or disables the dynamic test of exceeded axis limit.
What distinguishes the dynamic test of exceeded axis limit from the static test of exceeded axis limit is that the first one verifies at each real-time that the axis exceeds its limits, according to its current speed rate and to its maximum 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 integer variable or constant. It is the wait time limit, expressed in real-time.

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 real-time when the movement is executed, system error n. 3, "Servoerror", 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 real-times or if the movement ends in less than 200 real-times.

Example

```
; axis starting timeout equal to 10 real-times
ENABLESTARTCONTROL    x, 10
```

NOTCHFILTER

Syntax

NOTCHFILTER **axis, [value]**

Arguments

axis name of axis device
value constant or variable. Frequency value [Hz]. Valid values are in the range 0 to 500.

Description

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.

Example

```
; 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, status [value]**

Arguments

axis	name of axis device
status	default 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 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**, regardless of the activation or deactivation of the automatic adjust. The use of this instruction allows 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.

SETBACKLASH

Syntax

SETBACKLASH	**axis, value**

Arguments

axis	name of the axis device
value	variable or float constant. Backlash value

Description
This instruction allows 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.

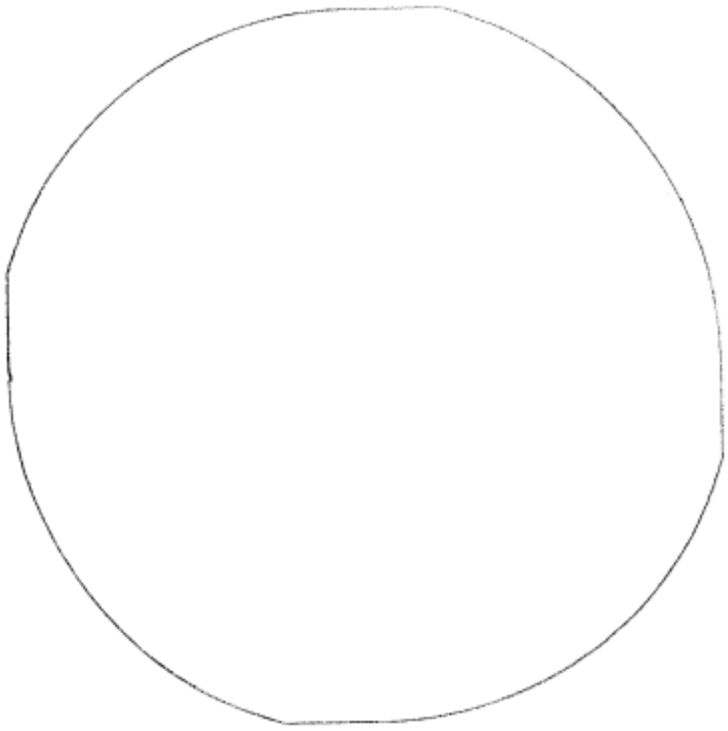
- In case of axis with a load of great inertia, there may be a partial or at times a total load compensation. As a matter of fact, due to the mass of the load, the motion of the axes could stop later than the engine. The resulting positioning of the reduction gear teeth as regards the teeth positioning 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 stepper, not controlled by TRS-AX remotes, counting, virtual axes
- on stepper axes, controlled by TRS-AX remotes with simulated encoder

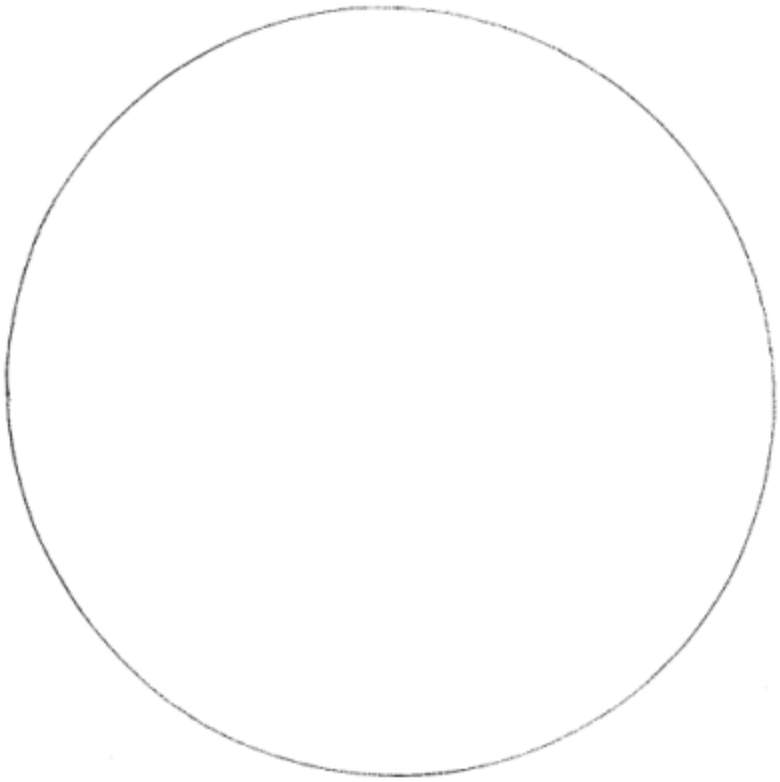
Example

```
; 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 higher threshold

Description

This instruction allows to modify the multiplication factor for the calculation of the higher threshold on **the axis** requested. To calculate the higher threshold, we need to multiply the variable **value** by the parameter defined in the axes configuration of the position arrival threshold. 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. Allowed 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, allowed 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, status**

Arguments

axis name of axis device
status default constant. Allowed values are:
ON zero pulse status enabled
OFF zero pulse status 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 stepping motor.
This instruction is used for axes moved by stepping 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 the **position** parameter is omitted, the configuration negative limit is enabled.
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 the **position** parameter is omitted, the configuration positive limit is enabled.
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	default 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

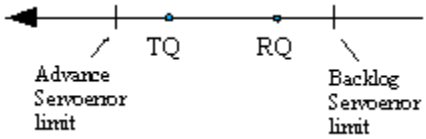
axis	name of axis device
backlog	constant or variable. Maximum backlog error
advance	constant or variable. Maximum advance error

Description

Sets the **axis** maximum values for the loop (**backlog)** and advance (**advance)** loop errors allowed by control, in negative direction, before generating "servoerror".

- **Backlog:** is the maximum tolerated loop value in the case of static test of the servoerror or in the case of dynamic servoerror test it is the value that, added to the theoretical error proportional to the speed, determines the maximum tolerated loop value.
- **Advance**: is the maximum loop value tolerated during the inversion of movement from negative movement to positive movement.

Loop error is computed as the difference between target position (where the axis should be positioned) and real position. 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

```
SETMAXERNEG    Axes.X, 10, 5
; Maximum axis delay is 10mm, maximum advance 5mm
```

SETMAXERPOS

Syntax

SETMAXERPOS **axis, backlog , advance**

Arguments

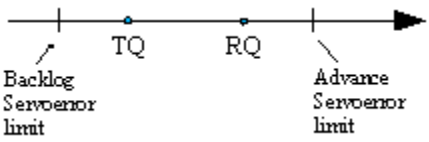
axis	name of axis device
backlog	constant or variable. Maximum backlog error
advance	constant or variable. Maximum advance error

Description

Sets the **axis** maximum values for the loop (**backlog)** and advance (**advance)** loop errors allowed by control, in negative direction, before generating "servoerror".

- **Backlog:** is the maximum tolerated loop value in the case of static test of the servoerror or in the case of dynamic servoerror test it is the value that, added to the theoretical error proportional to the speed, determines the maximum tolerated loop value.
- **Advance**: is the maximum loop value tolerated during the inversion of movement from negative movement to positive movement.

Loop error is computed as the difference between target position (where the axis should be positioned) and real position. 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

```
SETMAXERPOS    Axes.X, 10, 5
; Maximum axis delay is 10mm, maximum advance 5mm
```

SETMAXERTYPE

Syntax

SETMAXERTYPE **axis, type**

Arguments

axis name of axis device
type integer constant. Allowed values:
0 = sets servoerror to threshold value (default value)
1 = sets dynamic servoerror

Description

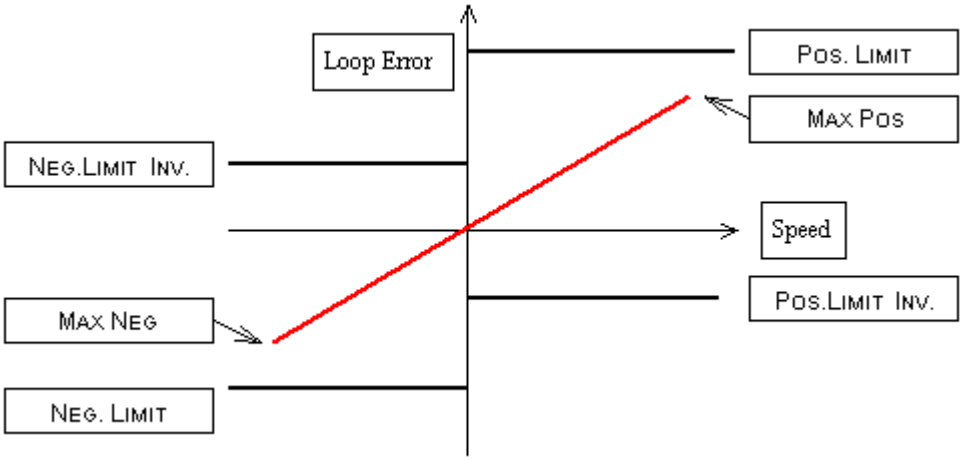
This instruction allows the **type** of servoerror test to be set. Conventional servoerror management sets a pair of limits (positive and negative), which are constant as axis speed changes. This type of management sizes the limits depending on the axis's maximum speed, i.e. it sets a limit so that the 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.

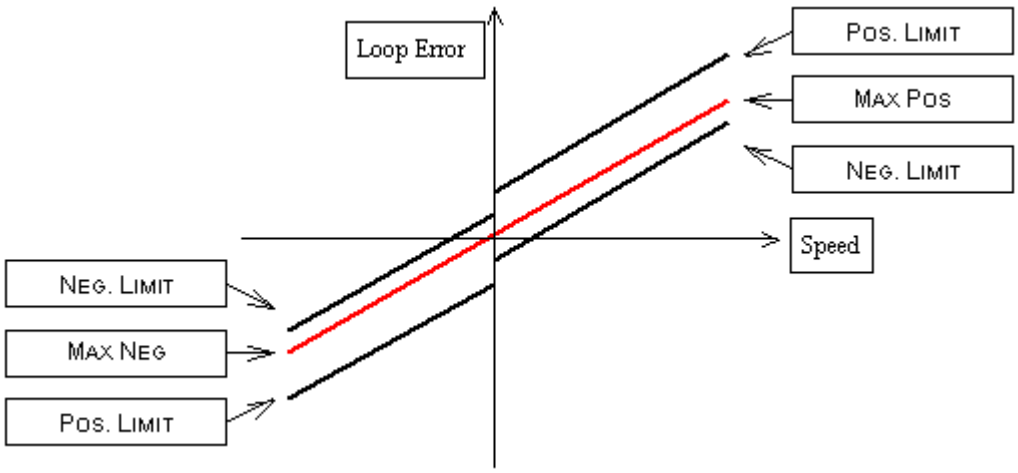
Nota

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.

Threshold ServoError Limit:



Dynamic ServoError Limit:



SETPHASESINV

Syntax

SETPHASESINV **axis, status**

Arguments

axis name of axis device

status	default constant. Allowed values: **ON** phase inversion stage enabled **OFF** phase inversion status disabled

Description

It enables or disables phases inversion on the indicated **axis**, allowing any encoder phase cabling inversion to be offset using software. If used with the reference inversion, the axis direction can be inverted (if cabling is correct).
To execute this instruction, the axis must be in a FREE status.

SETREFINV

Syntax

SETREFINV **axis, status**

Arguments

axis	name of axis device
status	default constant. Allowed values: **ON** reference inversion status enabled **OFF** reference inversion status disabled

Description

It enables or disables reference inversion on the indicated **axis.** If used with phases inversion, the axis direction can be inverted (if cabling is correct).
To execute this instruction, the axis must be in a FREE status.
See also SETPHASESINV.

SETRESOLUTION

Syntax

SETRESOLUTION **axis [, value]**

Arguments

axis	axis device name
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 status=coordinate), otherwise the system error no. 4101 "Inconsistent management of axis" is generated.

10.3.5 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.

10.3.6 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
; I set the timer TimeOut
; at a value of 20 seconds
SETTIMER    Timeout,20
; the timer starts in decreasing mode. When it gets to 0, it stops

STARTTIMER  Timeout,DOWN
```

STARTTIMER

Syntax

STARTTIMER	**timername [, direction]**

Arguments

timername	name of timer device
direction	default constant. Possible values are: **UP** increasing **DOWN** decreasing

Description

It starts the **timername** timer on the mode specified by **direction**, if specified.
If the **direction** parameter is omitted, **DOWN** mode is activated.
When a timer (started in decreasing mode) reaches zero, it automatically stops.
See also HOLDTIMER and SETTIMER.

10.3.7 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 **variable**.
If the value is not found, **variable** will get 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 **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 (compulsory argument). Number of elements of the vector
[n° rows]	constant or variable (compulsory 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 (compulsory argument)
mataddrname	name of addressee matrix
rowaddr	start rows number for the copy into the destination matrix (compulsory 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          ; it copies matrix Mx1 to Mx2

; it copies row 10 of matrix Mx1 to row 3 of Mx2
Movemat   Mx1[10], Mx2[3]

; it copies row 1 of matrix Mx1 to row 7 of Mx1
Movemat   Mx1[1], Mx1[7]
; it copies 6 rows starting from row 2 of matrix Mx1 to matrix
; Mx2, starting from row 8
Movemat   Mx1[2], Mx2[8],6
```

```
; it copies 4 rows starting from row 2
; of matrix Mx1 in the same matrix
; Mx1, starting from row 10
Movemat   Mx1[2], Mx1[10],4
```

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

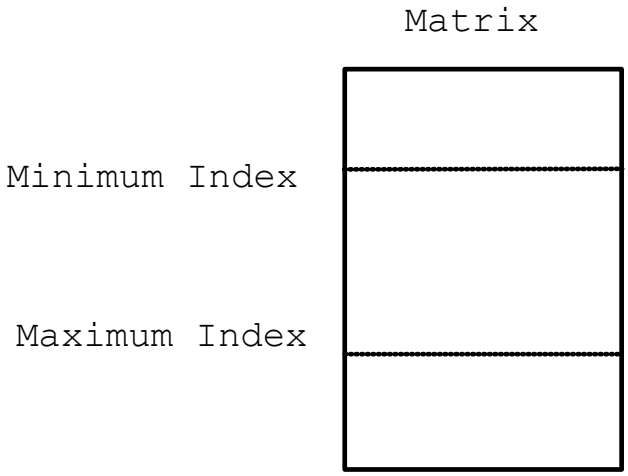
SORT	**matrix, column [, order], min_limit, max_ limit**
SORT	**vector [,order], min_limit, max_limit**

Arguments

matrix	name of the matrix
vector	name of the vector
column	constant or integer variable or countername. Matrix column number
order	default 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 indec of the vector or matric 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.



10.3.8 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

10.3.9 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 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. It is the vector where the read characters are stored
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**. The field **numchartoread** 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 **numcharread**. 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 **numcharread** 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 in the buffer. 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 is 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 to specify where the request for information is directed to. There are three classes of recipients:
- 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 cannot 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 specifying 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 status 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.
The following commands are allowed (Parameter **identifier**):
- Sim,0,container: it requires the Simulation button status, 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 status, 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 status, 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 receiving information about the status 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=manufacturer, 3=tpa
- "MascConfModules" 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"

The **container** parameter will receive the date and the time of the PC in a format relating to its type:
- char: number of the day of week
- integer: number of seconds from 1/1/1970
- float: number of days and fractions of a day from 1/1/1900
- double: number of days and fractions of a day from 1/1/1900
- string: text "AAAA/MM/GG hh:mm:ss"

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 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.xsyn*
- "Close:*filename*" closure of the synoptic *filename.xsyn*
- "*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.

"@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 reads 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 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

"@FileLastWrite"

Gets the date of the last modification made to a file. The **identifier** parameter is the name of the file (complete path) If in the identifier parameter the full path in which to store the file is not specified, Albatros considers the path defined in tpa.ini, under [tpa], then dirreport. The name of the file can be defined according to the rules described at the @FileRead **source** parameter. The **container** parameter will contain the date of the last modification to the file a format that is related to the type of the parameter:

- char: number of the day of week
- integer: number of seconds since January 1st,1970
- float: number of days and fractions of a days since January 1st, 1900
- double: number of days and fractions of a day since January 1st, 1900
- string: text in the format "YYYY/MM/DD hh:mm:ss"

"@FileInfo"

It read some info from a file. The parameter **identifier** must be expressed in the format "property:filename", where **property** stands for the name of the property to be read and **filename** is the name of the file. The name of the file can be set through "Name". The parameter **container** will contain the data read from the file.
List of properties:
- "version:": it returns in container an integer data. The four numbers identifying the version are in the 4 bytes of the container variable. Should an error occur, the value of the container variable is 0.
- "size:": it returns in container an integer, float, or double data. The data is the size of the file. Should an error occur, the value of the container variable is -1.

"@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#axis_name#HasFocus"	the parameter **container** contains 1, if the specified move axis window is active, otherwise it contains 0.
"MoveAX#axis_name#Jog"	the parameter **container** contains 1, if the move for displacements managed in runtime by the operator is set, otherwise it contains 0.
"MoveAX#axis_name#Step"	the parameter **container** contains 1, if the move with predefined steps is set, otherwise it contains 0.
"MoveAX#axis_name#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 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 displaying the "message box" on the screen and close down Albatros. Possible values for the **identifier** parameter are:

"Quit"	it shuts Albatros
"IsLocked"	it 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"	it reads the answer of a message box previously opened by a SEND

The parameter **container** makes it possible to know, in case of a message box, which button has been pressed by the operator:
- 1 "OK" button
- 2 "Cancel" button
- 4 "Retry" button
- 6 "Yes" button
- 7 "No" button

In case of the "Quit" control, the parameter **container** will contain the value:
- 1 if Albatros was shut correctly
- 0 if not

"@Param"

It allows knowing the progressive number of Partec.xpar and Partool.xpar parametric files storing. Requested information is stored into **container** parameter. Admited values for the parameter **identifier** are:

- "partec" it requests the progressive of partec.xpar storing
- "partool" it requests the progressive of partool.xpar 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"

It 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"

It executes the program defined in the parameter **identifier**. It 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 of the 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.

"@TermProg"

It 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.

"@ProgRunning"

It 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 of the program is also searched 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.

"@DialogFile"

It opens the dialog window of Open File or Save File to allow choosing the name of a file. To open the Open File window, set the parameter **identifier** = "Open", to open the window of Save File, set the parameter **identifier** = "Save". The choosen name of the file will be stored in the parameter **container**.

"@AxisCorrector"

It replaces the linearity corrector table of an axis with a new table, loaded from file, that anyhow shall have the same number of correctors and the crossed axes for corrector.
The parameter **identifier** is the name of the file, whose extention is typically .csv and is in the folder … \Mod.*n*\Config (the name of the file is 'normalized', as it happens, for instance, with "@FileExist").
The parameter **container** is defined as an integer variable and will contain 1, if the new correctors were sent, 0 otherwise.

"@Language"

It receives the translatable text corresponding to a group, library, or module message, that is associated to a MESSAGE instruction or an ERROR instruction. The values allowed for the parameter **identifier** are:

- "DEFMSG:number", where number is a sequence of digits. Albatros writes in **container** the text of the module message number "number".
- "DEFMSG:name", where "name" is the name, group and library included, of a DEFMSG. Albatros writes in **container** the text of the indicated message. If the group or library name is missing, the name of the task that sent the RECEIVE is used.
- "DEFMSG:*", Albatros writes in **container** the text of the group or module message previously indicated through the instruction SEND.

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

; it opens the window of File Open and stores the name of file in the
FileName variable
RECEIVE "@DialogFile", "Open", 0, FileName

; complete reading of a file
Function ReadProperties
 PARAM file     AS STRING
 LOCAL version  AS INTEGER
 LOCAL size     AS DOUBLE

 SEND “@FileName” “theFile” 0 file
 WAITRECEIVE “@FileInfo”, “version:theFile”, 0,version
 WAITRECEIVE “@FileInfo”,“size:theFile”, 0, size
```

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 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 be deleted only by the sender
$1000H	UpdateFlags	Modifies the status of the information (read / to be 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 status of the Simulating switch flag. According to the flag status, its identification button is visualized pressed or released in the toolbar (1=checked, 0=unchecked).
- Setp: notifies the change in status of the CmdSetp switch flag. According to the flag status, its identification button is visualized pressed or released in the toolbar (1=checked, 0=unchecked).
- Esc: notifies the change in status of the Excluded switch flag. According to the flag status, 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 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.xsyn*
- "Close:*filename"* closure of the synoptic *filename.xsyn*
- "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 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", it will display "11:29:40 PM".

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 consists of "ddd', 'MMM dd yy", it will display "Wed, August 31 94".
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** @FileRead.

"@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 get the following values:
- "Open" open Diagnostic
- "Close" close Diagnostic

When we want to interact with the axis movement window, the parameter **identifier** can get the following values:

"MoveAX#nome_asse#Open"	opening of the axis movement window
"MoveAX#nome_asse#Close"	closing of the axis movement window
"MoveAX#nome_asse#Plus"	pushing of the axis movement button (positive direction)
"MoveAX#nome_asse#Minus"	pushing of the axis movement button (negative direction)
"MoveAX#nome_asse#Stop"	pushing of the stop button

"MoveAX#nome_asse#Jog"	it sets the movement mode for displacements managed in runtime by the operator
"MoveAX#nome_asse#Step"	it sets the movement mode for displacements with predefined step
"MoveAX#nome_asse#Absolute"	it sets the movement mode with displacement of axis to a 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 closing Albatros or displaying "message boxes" on the screen, informing the user or asking permission for later actions. 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 a sequence of the following characters (they can indifferently be uppercase or lowercase):

"O"	"OK" button
"C"	"Cancel" button
"Y"	"Yes" button
"N"	"No" button
"R"	"Retry" button
"S"	Stop sign icon
"?"	Information icon, consisting of a lowercase 'i' within a circle
"!"	Exclamation mark icon
"*"	information icon
"1"	the first button is the default
"2"	the second button is the default
"3"	the third button is the default
"4"	the fourth button is the default

If not indicated, the default button is the first one.
For example "MsgBox:?YN2" identifies a message box with an information 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 TpaLangs.exe, or a group message label defined by the DEFMSG instruction.
As far as the text is concerned, if in it there is the newline character, "\u000A", the text will be split in two parts, and the first part will be displayed as text in the message box, while the second part will be displayed as explanation, or detail, of the text.
The language in which the two buttons will be displayed is the one of Windows.

"@Help"

opens a help file. It allows 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

"@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"
It 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"
it allows setting 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.

"@Language"
it sets the number of group, module, or library message that will be used in the following RECEIVE with the same identifier. The allowed value for the parameter **identifier** is "DEFMSG:*". The parameter **information** can be an integer variable or integer constant, and in this case it defines the number of group message to be displayed. It can be a string or a string constant, and in this case it defines the name of the DEFMSG.

Example

```
; Example of send file instruction with name created during execution.
: Supposing that the date of instruction execution
; is 31-01-2000

; in GPL
SEND "@File", "%Log%", 0, "Start execution"
```

```
; it adds a "Line Feed"
; in the tpa.ini file, section [TPA] we add
SEND "@File", "%Log%", 0
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
filename as string
; composition of the file name
setstring "C:\Albatros\report\LogFile.txt",filename
; association
SEND "@FileName", "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	default 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 allowing 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.

10.3.10 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 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 **variable** type.

The following operators are allowed:

()	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

Status of the port before executing the code

Status 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

```
Status of the port before code execution
```


```
Status 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

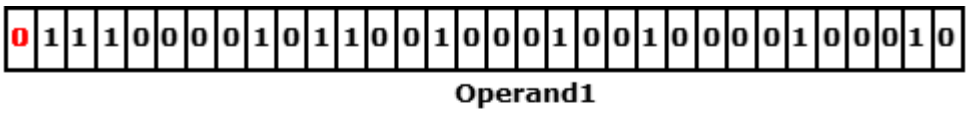
operand1 variable (integer or char) or name of device
operand2 variable (integer or char) or name of device

Description

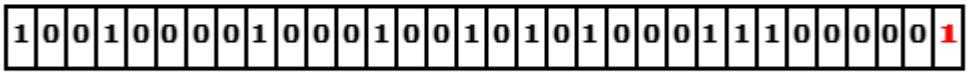
If **operand2** is not specified, this instruction performs a left hand shift operation of the bits that make up the **operand1.** If the second operand also is specified, a rotation is performed among the bits of **operand2** and the bits of **operand1**. At the end of the operation, **operand2** will contain the carry, that is the high bit of **operand1**.

Example

Integer operands rotation (Left hand shift with carry)
Before the rotation

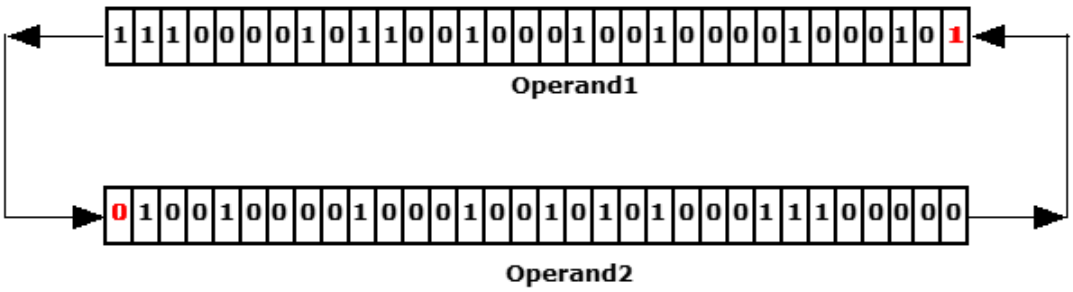


Operand1



Operand2

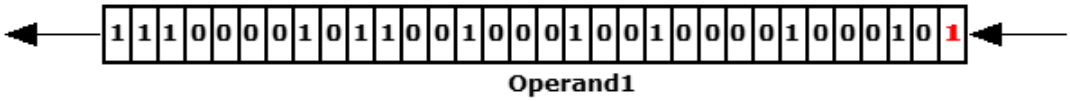
After the rotation



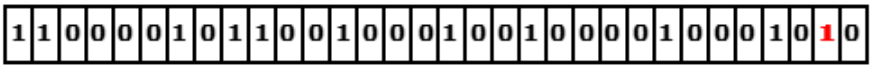
Operand1

Operand2

Shift (Left hand shift without carry)



Operand1



Operand1 after
shift to the left

SHIFTR

Syntax

SHIFTR **operand1 [, operand2]**

Arguments

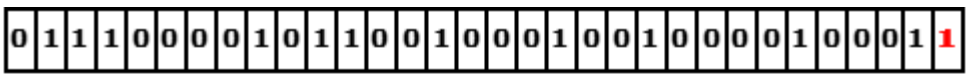
operand1 variable (integer or char) or name of device
operand2 variable or name of device

Description

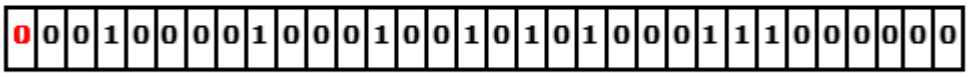
If **operand2** is not specified, this instruction performs a right hand shift of the bits that make up the **operand1**. If **operand1** is defined as char, the high entry bit is always 0. If **operand1** is defined as integer, the entry bit 32 is the mark bit.
If the second operand is also specified, it performs a rotation operation among the **operand2** defined as value 0 or different from 0 and the bits of **operand1**. At the end of the operation, **operand2** will contain the carry of the operation and the highest bit of **operand1** will be 0 or 1, according to the initial value of **operand2** (0 or 1).

Example

Integer operands rotation (Right hand shift with carry)
Before the rotation

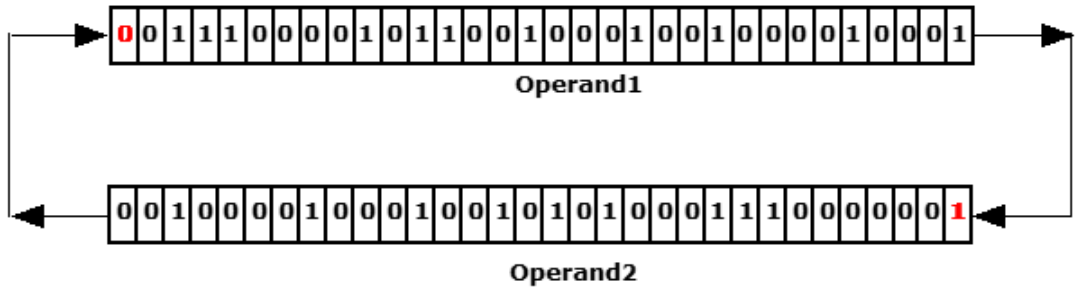


Operand1

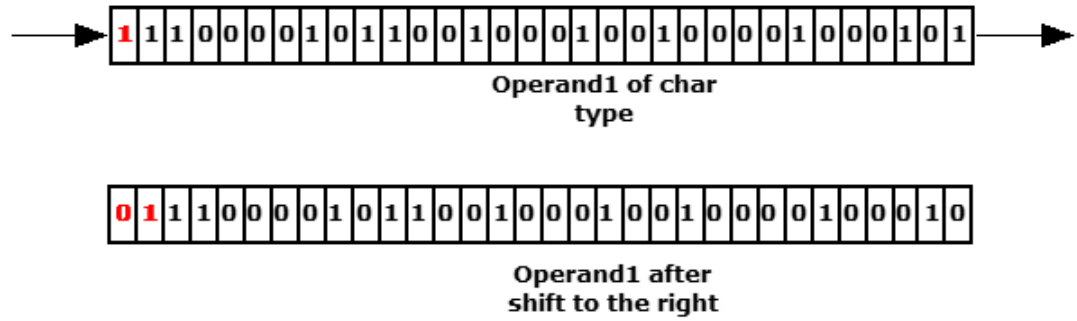


Operand2

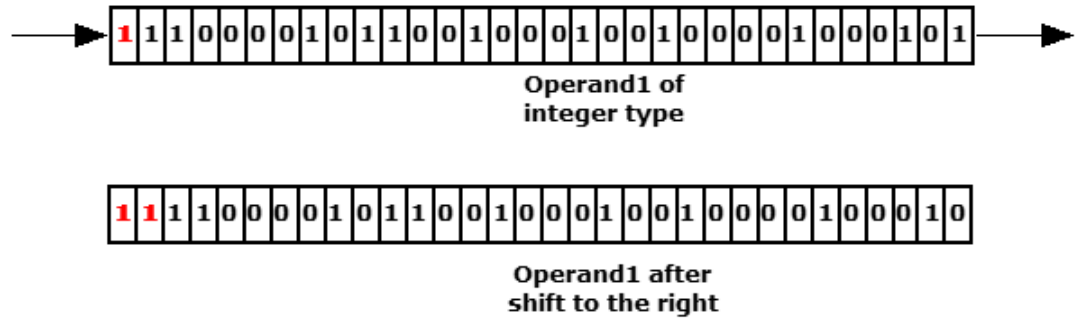
After the rotation



Char right hand shift (right hand shift without carry)



Integer 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)
```

10.3.11 Multitasking

ENDMAIL

Syntax

ENDMAIL **mail**

Arguments

mail constant or integer variable. Number of mailbox (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 gives back 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	default 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 (**varname** (1÷20) or the matrix row specified by **matrix[row]**) sent from the instruction is saved in a queue. In this case:

1. if the wait argument is **NOWAIT**, the execution carries on with the following instruction;
2. if the wait argument is **WAIT**, the 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 instruction ENDMAIL 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 assigns the priority value included in the **level** variable to 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.

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 real-time task are initialized only by the start of the task and then they maintain the value of the last run.

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

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 status, 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

10.3.12 Flow 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.

DELONFLAG

Syntax

DELONFLAG	**flagname**

Arguments

flagname name of flag device

Description

It disables the software interruption management on the status 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 status of an input which was previously enabled with the ONINPUT instruction.

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

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
Fret
```

IF/IFVALUE/IF-THEN-ELSE

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 synonyms. We suggest using the short version.
The instruction allows 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 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 is 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 further details, see the construct IF-THEN-ELSE.

IFBIT

Syntax

```
IFBIT                    mask, nbit, status, GOTO label
IFBIT                    mask, nbit, status, CALL subprogramname
IFBIT                    mask, nbit, status, functionname

IFBIT                    mask, nbit, status THEN
        instruction
        instruction
        ...
ENDIF

IFBIT                    mask, nbit, statuse 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)
status	predefined constant. Status 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 **status** is satisfied, it jumps to **label** or calls **subprogramname** or **functionname**.
For further details, see the construct 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 [, status], GOTO label**
IFCHANGEVEL	**axis [, status], CALL subprogramname**
IFCHANGEVEL	**axis [, status], functionname**

Arguments

axis	name of axis device
status	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 **status** 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: **<**ì (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 further details, see the construct 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 tests 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 further details, see the construct 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 further details, see the construct IF-THEN-ELSE.

IFERROR

Syntax

```
IFERROR                    number, IDposiz, GOTO label
IFERROR                    number, IDposiz, CALL label
IFERROR                    number, IDposiz, functionname
IFERROR                    devicename, status,IDposiz, GOTO label
IFERROR                    devicename, status,IDposiz, CALL label
IFERROR                    devicename, status,IDposiz, functionname
```

Arguments

number	DEFMSG or constant or integer variable
devicename	name of device
status	default constant. Acceptable values are: **ON**, **OFF**

IDposiz constant or variable. 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 the cycle error identified by **number** and **IDposiz** or by **devicename**, **status** and **IDposiz** is enabled, it can jump to **label** or call the function **functionname**.
The parameter **number** can identify an error of module cycle (therefore an entire numeric value) or of group (in this case a DEFMSG is used).
The parameter **devicename** is the name of a device and the parameter **status** represents the status ON/OFF in which the device is located, when the error is generated.
The parameter **number** can identify an error of module cycle (therefore an entire numeric value) or of group (in this case a DEFMSG is used).
The parameter **devicename** is the name of a device and the parameter **status** represents the ON/OFF status 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, status, GOTO label
IFFLAG                  flagname, status, CALL subprogramname
IFFLAG                  flagname, status, functionname

IFFLAG                  flagname, status THEN
        instruction
        instruction
        ...
ENDIF

IFFLAG                  flagname, status THEN
        instruction
        instruction
        ...
ELSE
        instruction
        instruction
        ...
ENDIF
```

Arguments

flagname name of flag device
status predefined constant. Status 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 status of a flag.
If the flag defined in the **flagname** variable satisfies the indicated **status**, it jumps to **label** or calls **subprogramname** or **functionname**.
For further details, see the construct IF-THEN-ELSE.

IFINPUT

Syntax

```
IFINPUT                 inputname, status, GOTO label
IFINPUT                 inputname, status, CALL subprogramname
IFINPUT                 inputname, status, functionname

IFINPUT                 inputname, status THEN
```

```
            instruction
            instruction
            ...
ENDIF

IFINPUT                 inputname, status THEN
            instruction
            instruction
            ...
ELSE
            instruction
            instruction
            ...
ENDIF
```

Arguments

inputname	name of input
status	predefined constant. Status 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 status of an input.
If the input specified in the **inputname** variable is in the indicated **status**, a jump to **label** or a **subprogramname** or **functionname** call is performed.
For further details, see the construct 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.

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 further details, see the construct IF-THEN-ELSE.

IFOUTPUT

Syntax

```
IFOUTPUT                outputname, status, GOTO label
IFOUTPUT                outputname, status, CALL subprogramname
IFOUTPUT                outputname, status, functionname

IFOUTPUT                outputname, status THEN
        instruction
        instruction
        ...
ENDIF

IFOUTPUT                outputname, status THEN
        instruction
        instruction
        ...
ELSE
        instruction
        instruction
        ...
ENDIF
```

Arguments

outputname	name of output
status	predefined constant. Status 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 status of an output.
If the input specified in the **outputname** variable is in the indicated **status**, a jump to **label** or a **subprogramname** or **functionname** call is performed.
For further details, see the construct 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 further details, see the construct 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 target 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 further details, see the construct 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 tests that the axis specified in the **axis** variable is in regime status.
If the condition is verified, 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 further details, see the construct 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 tests 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 status).
If the **nametask** is on 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 further details, see the construct 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 **value**, a jump to **label** or a **subprogramname** or **functionname** call is performed.
For further detail, see the construct 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 quiescent threshold (see Conventions and Terminology).
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 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 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 further details see the construct 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 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 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 the function execution, the task for which the error management was enabled stops, and it only restarts at the end of the first function called by the ONERRSYS instruction. In particular, the task will start again the execution by running again the instruction that was interrupted by the system error.

Example

Main Cycle with error management

ONFLAG

Syntax

ONFLAG **flagname, [status,] functionname[,arguments]**

Arguments

flagname	name of flag device
status	default constant. Status 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 status of the flag specified. When the flag switches to the indicated status (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 **status** argument is omitted, the function is called each time the status of the flag changes.
The test on the flag status 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, [status,] functionname [,arguments]**

Arguments

nameinput	name of input
status	predefined constant. Status 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 status of the input specified. When the input switches to the indicated status (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 are non-executable with interrupt.
If the **status** argument is omitted, the function is called each time the status of the input changes.
The test on the input status 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
instruction
CASE value1 TO value2
instruction
CASE IS < = > value
instruction
CASE ELSE
instruction
ENDSELECT

instruction it must be replaced by one of the following values:
GOTO label
CALL subprogramname
[FCALL] functionname [parameter1,...parameterN]
[EXPR] variable = expression
[EXPR] device = expression

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
variable	variable name
device	device name
expression	set of operators

Description

Multiple selection based on the **value** of the **varname** variable. The code in the CASE of the verified condition is executed. The CASE-ELSE branch is executed if no previous CASE is satisfied. For each CASE (optional) there can only be one GOTO, CALL or FCALL or EXPR 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 or EXPR, the execution of the function continues with the instruction after ENDSELECT.

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.

10.3.13 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	default constant. Language in which the message is written
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** (as for the list of language prefixes, please see chapter "Message Import"). 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.
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:

```
In a group:
DEFMSG MSG_TEST "Execution error"

FUNCTION CallTest
 Test MSG_TEST
FRET

In a library:
 DEFMSG MSG_BASE "Error signal: $1"
 …
FUNCTION Test Public
 PARAM code AS integer
 ERROR MSG_BASE NOPLACE NOSTORE code
FRET
; The displayed 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 [,status [, IDposiz [log]]**
DELERROR	**number [, IDposiz [log]]**

Arguments

devicename	name of the device
number	DEFMSG or constant or integer variable
IDposiz	constant or variable. It is a numeric value used in synoptics
status	predefined constant. It can have the following values: **ON**, **OFF**
log	predefined numerical constant or integer variable. It can have the following values: **STORE** error stored on file, **NOSTORE** error not saved on 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, then it sends the PC the type and logic address of the device instead of the number. For the clearance to be effective, all the values set in the parameters must coincide with those used to generate the error. Setting the parameter **log** as **STORE** entails storing the cycle error in the error report file of the current month. 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.
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 [,status [, 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.
status	predefined constant. Possible values are: **ON**, **OFF**
log	default numerical constant or integer variable. 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 the **log** parameter to **STORE** entails the cycle error being 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 100 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, CLEARERRORS.

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 ShowMessage
```

```
 MsgTool      MSG_ERR_CARICO MxTools[3].Cod
fret

Function ShowError
 ErrTool      STORE MSG_ERR_CARICO MxTools[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

Function ErrTool public
PARAM log as integer
PARAM argument1 as integer
PARAM argument2 as integer

 ERROR MSG_ERR_TOOL NOPLACE log argument1 argument2
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
comparison operator	the symbols used for comparison are: **<** (smaller than) **=** (equal to) **>** (greater than) **=<** (equal to or smaller than) **>=** (greater than or equal to) **<>** (different from)

value	constant or name of device
namegroup	name constant or name of group
devicename	string constant or device name

Description

The conditional compilation allows 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 does not 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 it is considered not connected.

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 MILL
; group is present in the machine
Const MillGroup = "Mill"
IFDEF Exist MillGroup
 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 3.2.0
; version of Albatros

IFDEF _VER_MAJOR = 3
 IFDEF _VER_MINOR = 2
   IFDEF _VER_REVISION = 0
     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 remote module
IFDEF _REMOTE_MODULE = 1        ; 1 = remote module, otherwise 0 = local
                                module
 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
 instruction
 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. The message is identified by the parameter **number**. The parameter **number** can identify a module (i.e. an integer number) or group message (in this case, DEFMSG applies. An argument, indicated by **IDposiz**, can also be optionally passed. 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 (IDposiz=0). The range of values that can be set is included between 0 (NOPLACE) and 1023.
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. The 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 100 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 MxTools[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 safety 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 the machine booting and has activated all emergency management tasks. At this stage the machine can be considered safe. You can define no more than 8 digital outputs. On each remote you can enable one only output. 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.

Note

The SYSYOK instruction can only be enabled:
- on all the GreenBus v3.0 remote modules with digital outputs;
- on the TRS-IO - v4.0-GreenBus remote modules;
- on the TRS-CAT remote modules, on the base only and not on the expansions, whose firmware version is 1.2 or higher (1.00 revision of the remote module)

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 to identify error situations occurring while executing the GPL code.
The first time this instruction is executed with the **status** parameter at ON, it enables the use of the watchdog.
The following times you always should assign ON to the **status** parameter to upgrade the counter of the board.
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

10.3.14 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 enabling 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 TPA.

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 AlbMech, DualMech and DualMech Mono boards. For further information about the use of this instruction contact TPA.

MECGETSTATUS

Sintax

MECGETSTATUS **axis,status,inout,error**

Arguments

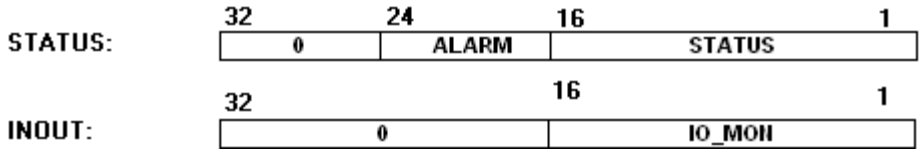
axis	name of digital axis device
status	constant or integer variable
inout	constant or integer variable
error	integer variable. Error code

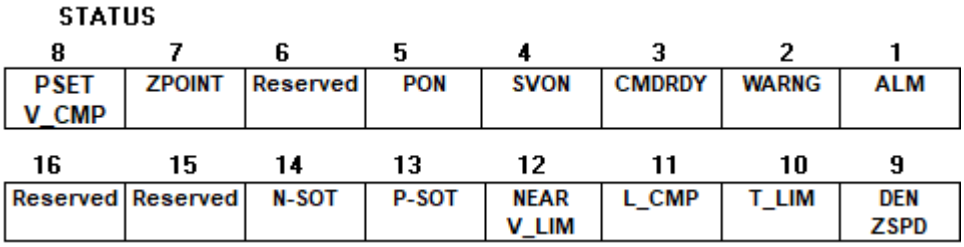
Description

It reads and stores the value of STATUS and ALARM into **status** variable and the value of IO_MON relative to specified **axis** into **inout** variable. 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 **status** and **inout** is a mask of bit structured as in the following representation:

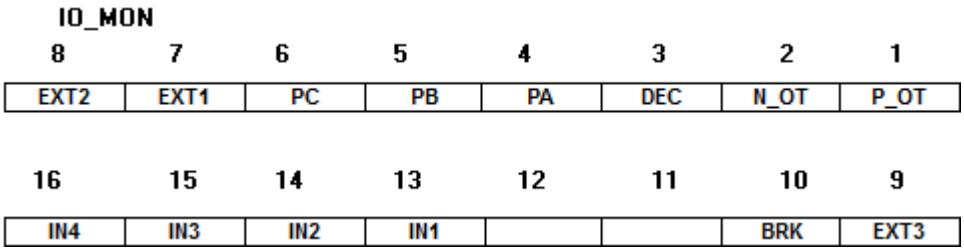




Meaning of STATUS bits

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) V_CMP (Velocity Agreement)	
9	DEN (Command Distribution Completed Flag) ZSPD (Zero Velocity)	
10	T_LIM (Torque Limit)	
11	L_CMP (Latch Completed)	
12	NEAR (Position Proximity) V_LIM (Velocity Limit)	
13	P-SOT (Forward-direction Software Limit)	
14	N-SOT (Reverse-direction Software Limit)	
15	Rerserved	
16	Reserved	



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 DualMech Mono boards. For further information about the use of this instruction contact TPA.

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 TPA.

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.

10.3.15 Standard fieldbus

AXCONTROL

Syntax

AXCONTROL **axis, data**

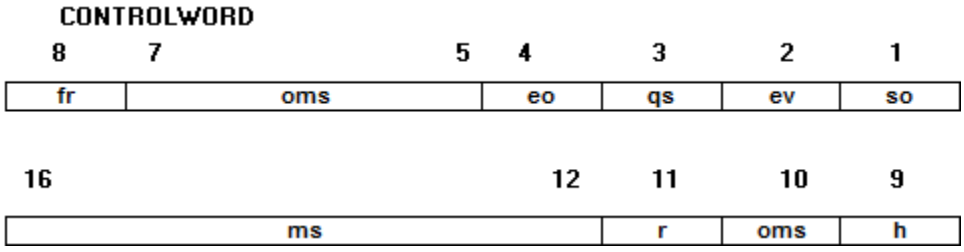
Arguments

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".

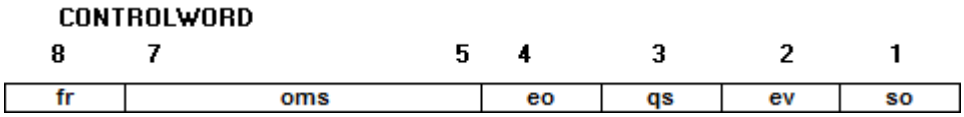
EtherCAT value definition table



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



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

AXSTATUS

Syntax

AXSTATUS **axis, value**

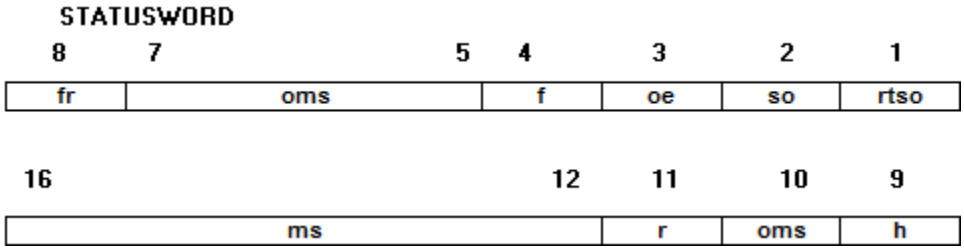
Arguments

axis device name of axis type
value integer variable

Description

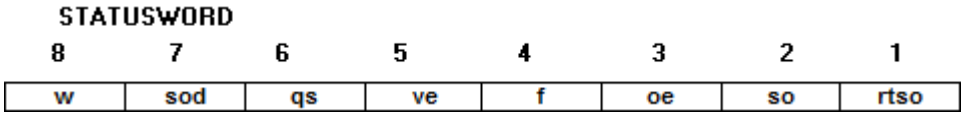
It return the value in the StatusWord in accordance with "CiA 402 CANopen device profile".

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 0:axis stall	SW2
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. Node number returned

Description

Returns board number and node number 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.
The instruction can be used for devices configured upon CAN, S-CAN, Greenbus or EtherCAT buses.
If the returned node number is a negative value, it means that the node is disabled.

READDICTIONARY

Syntax

READDICTIONARY **board,cn,index,subindex,dimdata,data,err**

Arguments

board constant or integer variable. Board number
cn constant or integer variable. Node 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,char vector,string. Variable receiving the data
err integer variable. Error code returned by node

Description

It reads the content of an objects' dictionary object, contained in node. 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 node 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 node manufacturer. For the S-CAN devices the sub-index parameter must always be set to zero.

WRITEDICTIONARY

Syntax

WRITEDICTIONARY **board,cn,index,subindex,dimdata,data,err**

Arguments

board constant or integer variable. Board number
cn constant or integer variable. Node 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,char vector,string. Variable containing the data
err integer variable. Error code returned by node

Description

It writes the content of an objects' dictionary object, contained in node. 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 node 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 node manufacturer. For the S-CAN devices the sub-index parameter must always be set to zero.

10.3.16 EtherCAT

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

ECATGETREGISTER

Syntax

ECATGETREGISTER **node, address, dim, data, error**

Arguments

node constant or integer variable. Position held by the slave in the EtherCAT chain (from 1 onwards)
address constant or integer variable. Address of the ESC register (EtherCAT Slave Controller) to read (from 0 onwards)
dim constant or integer variable. Number of bytes to read (1, 2 or 4)
data integer variable. Container of the read data
error variable integer. Error code

Description

Returns **[data]** the register contents of the ESC (EtherCAT Slave Controller) for the indicated EtherCAT node. The **error** parameter will contain the numerical code of the error, 0 if no errors have occurred.

ECATSETREGISTER

Syntax

ECATSETREGISTER **node, address, dim, data, error**

Arguments

node constant or integer variable. Position held by the slave in the EtherCAT chain (from 1 onwards)
address constant or integer variable. Address of the ESC register (EtherCAT Slave Controller) to write (from 0 onwards)
dim constant or integer variable. Number of bytes to write (1, 2 or 4)
data constant or integer variable. Data to be written
error variable integer. Error code

Description

This instruction writes the content **[data]** of a register of the ESC (EtherCAT Slave Controller) for the indicated EtherCAT node. The **error** parameter will contain the numerical code of the error, 0 if no errors have occurred.

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 defined in the hardware configuration for the node in question (from 1 to 16)
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 32)
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.

SETEOE

Syntax

SETEOE **status,[error]**

Arguments

status	predefined constant. acceptable values are: - ON enables the management of the EoE protocol - OFF deactivates the management of the EoE protocol
error	variable integer. Error code

Description

This instruction allows to activate and deactivate the management of the EoE protocol (Ethernet over EtherCAT). This functionality uses the SnifferCE or SnifferRTX auxiliary software managing the EoE communication between numerical control and PC supervisor. The error variable can accept following values:
0 if Albatros command was properly executed
-1 if the command has not been terminated within the maximum time of 5 seconds
-2 if a communication error occurred

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 defined in the hardware configuration for the node in question (from 1 to 16)
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 32)
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.

10.3.17 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** as shown. For further information on the meaning of these parameters, please the documentation of the single 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** as shown. For further information on the meaning of these parameters, please read the documentation of the single 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 **board** as shown. For further information on the meaning of these parameters, please read the documentation of the single device.

RECEIVEPDO

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 **PDOnumber** for the mentioned node. This instruction is used to read the asynchronous PDOs (i.e. those PDOs that in the hardware configuration have the **Asynchronous** option activated).
The read data are copied in the devices connected to the virtual-physical device.
This instruction can only be used with TMSCan and TMSCan+ boards.

SENDPDO

Syntax

SENDPDO **board, node, PDOnumber**

Arguments

board constant or variable integer. Number of the board
node constant or variable integer. Number of the node
PDOnumber constant or variable integer. PDO number

Description

It writes the specified PDO content from **PDOnumber** for the mentioned node. This instruction is used to write the asynchronous PDOs (i.e. those PDOs that in the hardware configuration have the **Asynchronous** option enabled).
This instruction can only be used with TMSCan and TMSCan+ boards.

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** 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 the channel. For further information about the meaning of these parameters, make reference directly the documentation concerning each single device.

Value	Protocol status
1	Operational
128	Pre-Operational

10.3.18 Simulation

DISABLE

Syntax

DISABLE **axis1,[...axis6]**

Arguments

axis1...[...axis6] name of axis devices

Description

It disables the specified axes. This allows 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 board during simulated movements, that is when axes are disabled. See also ENABLE.

Note

Stepper 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 status. See also ENABLEFORCEDINPUT, DISABLEFORCEDINPUT, SETFORCEDINPUT,RESETFORCEDINPUT, 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

Stepper 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, 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, 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, SETFORCEDPORT.

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, 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, SETFORCEDANALOG.

10.3.19 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 statuses of the stored devices. The database has a table containing a temporal information and the status of all devices in that time, one for each column. In the GPL language new instructions have been introduces to start, 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 TPA).

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 TPA).

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 status of the logic devices excluding the flag switch.
Recording period (**value**) is expressed in milliseconds. It cannot be less than 10 and it must be a multiple of the real-time period. Otherwise, the system error no. 4399 (Parameter out of 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	Could not open the database on disc
9	Could not create in the database the recording table
10	Error in IP address for the communication with the remote module (remote modules only)
11	Could not create the communication socket to receive the data (remote modules only)
12	Could not associate a local address to the communication socket (remote modules only)
13	Could not connect to the remote socket (remote modules only)
14	Could not 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.

10.3.20 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 stepper 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 management of axis AxisName" is generated.

ISOG1

Syntax

ISOG1 **label, axis1, position1,axis2, position2, axis3, position3,axis4,position4,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 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 rate 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 stepper 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 management of axis AxisName" 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 **rotaryMatrixName, toolHolderMatrixName, toolMatrixName, enablingMask, axis1, axis2, axis3, axis4, axis5**

Arguments

rotaryMatrixName	name of the matrix. It contains the data concerning the rotary axes.
toolHolderMatrixName	name of the matrix. It contains the data concerning the toolholder.
toolMatrixName	name of the matrix. It contains the data concerning the tools.
enablingMask	integer variable or constant. Enabling mask of C and B axes.
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 machine parametrization and the five axis devices the machine consists of. This instruction **must** be executed before any other ISO instruction. The parameter **enablingMask** defines which rotary axes (C and/or B) should be enabled.
As for the values to be set, please refer to the following table:

enablingMask	**Description**
31	Disabling of C and B axes
23	Enabling B axis only
15	Enabling C axis only

7 Disabling of C and B axes

Note
The unit of measure in which the rotary axis values are expressed in the configuration must be degrees.
The link among physical axes and virtual ISO axes set through this instruction is broken when the task where the instruction is defined is ended, or through the ISOM2 instruction. Therefore, the axes can be used for classic movements.

ISOG217

Syntax

ISOG217 **axis1,axis2,axis3,axis4,axis5,virtualAxis1,virtualAxis2,virtualAxis3, virtualAxis,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 stepper 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.

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
value 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 **value** 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 curvilinear 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	(Greater than 0.0)-1.0	0.2	Minimum dimension of the space to cover with only linear axes. The value is expressed in millimeters.
6	(Greater than 0.0)-1.0	0.1	Minimum dimension of the space to cover with only rotary axes. The value is expressed in degrees.
7	0.0-100.0	100.0	Smooth filter lower limit
8	1.0.100.0	1.0	Multiplication factor applied to the accelerations and to the decelerations defined in the configuration. It increments the acceleration and the max. deceleration of the only linear

			axes. Values external to the interval generate the system error no. 4399 – Parameter out of range.
9	1.0-100.0	1.0	Multiplication factor applied to the accelerations and to the decelerations defined in the configuration. It increments the acceleration and the max. deceleration of the only rotary axes. Values external to the interval generate the system error no. 4399 – Parameter out of range.
10	0.0-1.0	0.0	Flag to enable (0.0) or disable (1.0) the speed among consecutive blocks to be reduced in case of angular point. The deactivation can also be obtained by using the instruction with the following parameters: ISOSETPARAM 0 0.0 ISOSETPARAM 1 0.0 ISOSETPARAM 2 1.0

KINEMATICEXPR

Syntax

KINEMATICEXPR **axis = expression**

Arguments

axis name of device of physical or virtual axis type
expression group of operators

Description

It allows 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 operative 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
```

10.3.21 Instructions that are no longer available

INPBCD	reads a set of digital nibbles in BCD format
OUTBCD	modifies a set of digital nibbles in BCD format
SETFORCEDBCD	forces a nibble set in BCD format
CANOPENDRIVER	opens a CANopen communication channel
CANCLOSEDRIVER	closes a CANopen communication channel
CANRESETBOARD	resets a CANopen board
CANSETOBJECT	writes a CANopen object
CANGETOBJECT	reads a CANopen object
SLMCOMMAND	runs a SLM command
SLMEEPROMDISABLE	runs an EEPROM writing disabling command
SLMEEPROMENABLE	runs an EEPROM write permission
SLMGETEEPROM	reads an EEPROM memory location
SLMGETPARAM	reads a SLM parameter
SLMGETREGISTER	reads a SLM register
SLMGETSTATUS	reads a drive quantity
SLMSETEEPROM	writes a location of EEPROM memory
SLMSETPARAM	sets a SLM parameter
SLMSETREGISTER	sets a SLM register
HOMING	searches for the "zero position"

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
GETVF	reads the tension/frequency converter value

10.3.22 Instructions that cannot be used with interrupt

The following instructionscannot be used in functions called by ONFLAG, ONINPUT, and ONERRSYS. Their use is not allowed in real-time tasks either.

Instructions that, in turn, call a function on interrupt:
- ONFLAG
- ONINPUT
- ONERRSYS

Instructions that 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

The following instructions are pertinent to 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

Blackbox Instructions:

- ENDBLACKBOX
- PAUSEBLACKBOX
- STARTBLACKBOX

The following instructions are pertinent to EtherCAT management:

- READDICTIONARY
- WRITEDICTIONARY

Instructions pertinent to multitasking:

- SENDMAIL
- WAITMAIL
- ENDMAIL
- SENDIPC
- WAITIPC
- TESTMAIL
- TESTIPC

Instructions that imply a long processing time:

- SORT
- FIND
- FINDB
- MOVEMAT

10.4 Examples

10.4.1 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 status.
;        If it is on ON, it moves the axis and waitsfor it to return
;        to OFF status. 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
;
;
;  © TPA
;--------------------------------------------------------------------
Function Fast_Homing

   ResLimPos   axis          ; Axis start-up
   ResLimNeg   axis
   SetQuote    axis,0

   IfInput     FastInput,OFF,Goto Continue  ; Tests occupied sensor
   SetVel      axis,5                       ; Sets the
                                            ; disengagement speed
   MovAbs      axis,30                      ; Moves the axis
   WaitInput   FastInput,OFF,30,Call Error  ; Tests the micro
                                            ; disengagement,
                                            ; Error after TimeOut=30
   EndMov      axis                         ; Stop of the axis
   WaitStill   axis                         ; waits while the axis stops

Continue:
   SetVel      axis,10                      ; Homing sensor search speed
   MovAbs      axis,-1000                   ; Sensor search negative movement
   SetPFly     axis,ON,10,0                 ; Interrupts hook
                                            ; and sets disengagement
                                            ; speed and position
   WaitStill   axis                         ; wait while the axis stops
   SetLimPos   axis                         ; Reset axis limits
   SetLimNeg   axis

   Fret

; subprocedure to send error messages
Error:
   Error       ERR_SETP                     ;Error signalled: impossible to
                                            proceed
   Ret
```

10.4.2 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 status 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     ; tool change
Const FORO = 12     ; executes hole


;--------------------
; --- AXIS 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 hole
   local position_Y as double        ; position Y hole

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                         ; Axis Homing
        fcall homing_axes
   case CHG                          ; Executes tool change
        fcall Change_tool
   case FORO                         ; hole in
        fcall Perforation            ; specified position
        position_X,position_Y

   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   ; tests flag status
```

```
   ; 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).
```

10.4.3 Main Cycle with error management

```
;-------------------------------------------------------------
; Hypothetical main function
; starts machine and executes test cycle
;-------------------------------------------------------------
Function Main
   OnErrSys        GestErrSys          ; enables 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
                                               ; are related to
                                               ; the axes
   ResetFlag       Flag
   Disableaxes

noerraxis:
Fret
```

10.4.4 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 for",string1       ; string1 now contains
                                         ; "String for"

  SetString   " test",string2

  AddString   string1,string2,string3    ; stringa3 contains
                                         ; "String for test"

  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 "st"

  Mid         string3,9,2,string3        ; string3
                                         ; contains "for"

  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
```

10.4.5 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          ; informs the operator

  Fcall       HomingAxisZ       ; Homing of Z axis

  ; OK Z axis Homing is concluded
```

```
   StartTask    HomingAxisX      ; launches the homing of X and Y
   StartTask    HomingAxisY

   WaitTask     HomingAxisX      ; wait for the task to end
   WaitTask     HomingAxisY

   DelMessage   MSG_SETP         ; deletes the message
                                 ; for the operator

Fret
```

10.4.6 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
;
;   © TPA.
;-----------------------------------------------------------------

Function Homing

   ResLimPos     axis          ; disables the software limits
   ResLimNeg     axis

   SetVel        axis,10       ; sets the speed

   MovInc        axis,10000    ; moves the axis

   WaitInput     Switch,ON     ; waits for the switch

   EndMov        axis          ; stops the axis
   waitStill     axis          ; waits for the axis to stop

   SetVel        axis, 0.1     ; sets the disengagement speed

   MovInc        axis,-100     ; moves the axis

   WaitInput     Switch,OFF    ; waits for the switch
                               ; to disengage
   SetQuote      axis,0        ; assigns the new position

   SetVel        axis          ; resets the speed
   SetLimpos     axis          ; resets the software limits
   SetLimneg     axis

Fret
```

10.4.7 Iso movements

```
-----------------------------------------------------------------
;  Example of ISO movement
;
;  A profile is generated using the instruction ISOG0 and ISOG1
;
```

```
;  © TPA.
;-------------------------------------------------------------------*

; 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
; Tool 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 srl

Via Carducci 221
I - 20099 Sesto S.Giovanni
(MI)
Tel. +39 02.36527550
Fax. +39 02.2481008
www.tpaspa.com