

UltimET Light Motion Controllers

User's Manual

Version T

ETEL

THIS PAGE IS INTENTIONALLY LEFT BLANK

Table of contents

Chapter A: Internal functioning & generalities

| | | |
|----------|---------------------------------------------------------|-----------|
| 1 | UltimET Light principle | 14 |
| 2 | UltimET Light internal functioning | 15 |
| 2.1 | Communication with UltimET Light | 15 |
| 2.2 | Several commands types | 15 |
| 2.3 | Block diagram | 15 |
| 2.3.1 | Communication | 15 |
| 2.3.2 | Several command sources | 16 |
| 2.3.3 | Management of the interpolation commands | 17 |
| 2.3.4 | Management of a mix of normal and interpolated commands | 19 |
| 2.3.5 | Multi-thread sequences | 21 |
| 2.4 | Ordering information | 22 |

Chapter B: Installation and setting

| | | |
|----------|-------------------------------------------------|-----------|
| 3 | Initial installation | 24 |
| 3.1 | UltimET Light PCI | 24 |
| 3.1.1 | Hardware characteristics | 24 |
| 3.1.2 | Electrical interface | 24 |
| 3.1.3 | Software characteristics | 26 |
| 3.1.4 | LEDs meaning | 28 |
| 3.2 | UltimET Light PCIe | 29 |
| 3.2.1 | Hardware characteristics | 29 |
| 3.2.2 | Electrical interface | 29 |
| 3.2.3 | Software characteristics | 29 |
| 3.2.4 | LEDs meaning | 30 |
| 3.3 | UltimET Light TCP/IP for AccurET optional board | 30 |
| 3.3.1 | Hardware characteristics | 30 |
| 3.3.2 | Electrical interface | 31 |
| 3.3.3 | Hardware installation | 32 |
| 3.3.4 | Software characteristics | 32 |
| 3.3.5 | LEDs meaning | 38 |

Chapter C: UltimET Light commands

| | | |
|----------|----------------------------------------|-----------|
| 4 | Commands & registers syntax | 40 |
| 4.1 | Commands | 40 |
| 4.2 | Registers | 40 |

| | | |
|--------|-------------------------------------------------|----|
| 4.2.1 | Registers group | 40 |
| 4.2.2 | Register value reading | 42 |
| 4.2.3 | Register value writing | 42 |
| 4.2.4 | CLX command | 44 |
| 4.2.5 | SET_RANGE command | 45 |
| 4.2.6 | CHKDISTGRT command | 45 |
| 4.3 | Communication with the other nodes | 45 |
| 4.3.1 | Command for another node | 45 |
| 4.3.2 | Parameter coming from another node | 46 |
| 4.4 | Save the settings | 46 |
| 4.5 | UltimET Light reset | 48 |
| 4.6 | UltimET Light software characteristics | 48 |
| 4.7 | Synchronized homing on several axes | 49 |
| 4.8 | WAIT command | 54 |
| 4.8.1 | Wait command list | 54 |
| 4.9 | User LEDs | 57 |
| 4.10 | Local digital inputs / outputs | 57 |
| 4.10.1 | Digital inputs | 57 |
| 4.10.2 | Digital outputs | 58 |
| 4.11 | External inputs / outputs | 59 |
| 4.11.1 | Configuration | 60 |
| 4.11.2 | Use and setting of external I/Os on WAGO | 61 |
| 4.12 | Traces management | 68 |
| 4.12.1 | Traces configuration | 68 |
| 4.12.2 | Non-synchronized mode | 70 |
| 4.12.3 | Synchronized mode | 71 |
| 4.12.4 | Trigger condition test and edge detection | 72 |
| 4.12.5 | Traces upload | 72 |
| 4.12.6 | Continuous traces | 72 |
| 4.13 | Errors and warnings management | 73 |
| 4.13.1 | Errors propagation by the UltimET | 73 |
| 4.13.2 | ERR command | 74 |
| 4.13.3 | Errors propagation on the TransnET | 74 |
| 4.13.4 | TransnET functioning in case of error | 75 |
| 4.13.5 | Errors reset | 76 |
| 4.13.6 | Warnings | 76 |
| 4.14 | Status | 77 |
| 4.14.1 | UltimET Light controller status | 77 |
| 4.14.2 | 'Sequence' bit of the UltimET Light | 77 |
| 4.14.3 | 'Interrupt' bit | 78 |

| | |
|----------------------------------------------------------------------------------------|-----------|
| 4.15 Set / reset bit(s) management | 78 |
| 4.16 Set several registers | 78 |
| 5 Real-time channels on TransnET | 79 |
| 5.1 Principle | 79 |
| 5.2 Data sharing | 79 |
| 5.2.1 Between controllers | 79 |
| 5.2.2 Between controllers and the PC application (UltimET Light PCI / PCIe only) | 79 |
| 5.2.3 Disappearing of TransnET | 80 |
| 5.3 Interface description | 80 |
| 5.3.1 Slot management | 81 |
| 5.3.2 UltimET – AccurET functions for sharing data | 81 |
| 5.3.3 Real-time channels timing | 84 |
| 5.3.4 Forward command | 84 |

Chapter D: Interpolation commands

| | |
|-----------------------------------------------------------------|-----------|
| 6 Interpolation | 88 |
| 6.1 Introduction | 88 |
| 6.1.1 Units of the interpolation commands | 88 |
| 6.1.2 Distance and time limits for interpolation commands | 89 |
| 6.2 Setting of the interpolation | 90 |
| 6.2.1 ISET command | 90 |
| 6.2.2 IBEGIN command | 91 |
| 6.2.3 IEND command | 97 |
| 6.2.4 Two groups of interpolation | 97 |
| 6.3 User position vs. real position | 98 |
| 6.4 Interpolation movements | 98 |
| 6.4.1 'G-code based' mode | 98 |
| 6.4.2 PVT (Position-Velocity-Time) mode | 108 |
| 6.4.3 PT (Position-Time) mode | 114 |
| 6.4.4 ULM mode | 116 |
| 6.4.5 Braking commands | 122 |
| 6.4.6 Time commands | 124 |
| 6.4.7 Example of time treatment of interpolated commands | 126 |
| 6.4.8 Tangential speed | 127 |
| 6.5 'Moving' bits | 128 |
| 6.6 Marks | 128 |
| 6.6.1 Identification of the mark | 128 |
| 6.6.2 Action associated with a mark | 128 |
| 6.7 Matrix | 131 |

| | | |
|-------|----------------------------|-----|
| 6.7.1 | Translation function | 131 |
| 6.7.2 | Scaling function | 132 |
| 6.7.3 | Rotation function | 133 |
| 6.7.4 | Shearing function | 134 |
| 6.7.5 | Matrix restore | 135 |
| 6.8 | Conversion factor | 135 |

Chapter E: Programming

| | | |
|--------|------------------------------------------------------------------------------|-----|
| 7 | Programming of the controller | 138 |
| 7.1 | Introduction | 138 |
| 7.2 | Basic concepts | 138 |
| 7.2.1 | Source file | 138 |
| 7.2.2 | Comments | 138 |
| 7.2.3 | Header | 139 |
| 7.2.4 | Register and command | 139 |
| 7.2.5 | Instruction delimiter | 139 |
| 7.2.6 | Immediate values | 139 |
| 7.2.7 | C preprocessor define | 140 |
| 7.2.8 | Thread | 140 |
| 7.2.9 | User variables X | 140 |
| 7.2.10 | Advanced management of the X variables in a multi-thread configuration | 141 |
| 7.3 | Structured code | 143 |
| 7.3.1 | Identifiers | 143 |
| 7.3.2 | Types | 144 |
| 7.3.3 | Global variables | 144 |
| 7.3.4 | Registers | 145 |
| 7.3.5 | Functions | 145 |
| 7.3.6 | Operators | 147 |
| 7.3.7 | Expressions | 150 |
| 7.3.8 | Statements | 151 |
| 7.3.9 | Unit conversion | 156 |
| 7.3.10 | Predefined functions | 157 |
| 7.3.11 | Sequence gates | 158 |
| 7.3.12 | Inputs / outputs management | 160 |
| 7.4 | How to start a sequence | 162 |
| 7.4.1 | Start a sequence thread from ComET | 162 |
| 7.4.2 | Start a sequence thread from an application running on the PC | 163 |
| 7.4.3 | Start a sequence on the AccurET from a sequence present in the UltimET | 163 |
| 7.4.4 | Start a sequence thread from another thread | 163 |
| 7.4.5 | Start a sequence thread automatically | 163 |
| 7.5 | How to stop a sequence thread | 163 |

| | |
|---------------------------------------------------|------------|
| 7.6 Error management | 164 |
| 7.6.1 Predefined error routine | 165 |
| 7.6.2 ERR command | 165 |
| 7.7 Performance aspects | 165 |
| 7.7.1 Allocation of time to the compiled sequence | 165 |
| 7.7.2 Compiled sequence stack management | 166 |

Chapter F: Appendixes

| | |
|----------------------------------|------------|
| 8 Commands reference list | 168 |
| 9 Parameters *K | 183 |
| 10 Monitorings *M | 187 |
| 11 Warning reference list | 193 |
| 12 Errors reference list | 193 |
| 13 Service and support | 199 |

Index

THIS PAGE IS INTENTIONALLY LEFT BLANK

Record of revisions:

| Document revisions | | |
|--------------------|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Version | Date | Main modifications |
| Ver A | 15.05.09 | First version |
| Ver B | 23.10.09 | Updated version: - UtimET Light TCP/IP version added (refer to §3.3) - *IMARK command modified (refer to §6.6) - *IND command modified (refer to §4.7) - I/O functions added (refer to §4.10) - Indirect parametrization (refer to §4.2.3) |
| Ver C | 03.01.11 | Updated version: - New homing mode (refer to §4.7) - Error propagation on TransnET (refer to §4.13.3) - Real-time channels (refer to §5.) - Sequence gate (refer to §7.3.11) |
| Ver D | 12.03.12 | Updated version: - External inputs / outputs (refer to §4.11) - Error propagation monitoring (refer to §4.13.3) - Inputs / outputs management (refer to §7.3.12) |
| Ver E | 12.11.12 | Updated version (with firmware from version 2.06A): - New parameter *K724 for IO configuration (refer to §4.11.1.2) - Minor changes |
| Ver F | 17.04.13 | Updated version (with firmware from version 2.07A): - Warning management (refer to §4.13.6) - Management of the interpolation commands improved (refer to §2.3.3) - ASR and ERR commands improved (refer to §5.3.2.2 and §4.13.2) |
| Ver G | 05.05.14 | Updated version (with firmware from version 3.00A): - New commands: CLRWAIT (refer to §4.8.1.6), CH_BIT_REG32 (refer to §4.15) and SET_RANGE (refer to §4.2.5) - Command management description improved (refer to §2.3.4 and §6.4.7) - New parameter K144 (refer to §7.5) |
| Ver H | 17.12.14 | Updated version (with firmware from version 3.01A): - UtimET Light PCIe version added (refer to §3.2) - New command: SETRTV (refer to §5.3.1) - Classic code removed because not used any more |
| Ver I | 12.05.15 | Updated version (with firmware from version 3.02A) - Minor changes |
| Ver J | 13.07.16 | Updated version (with firmware from version 3.10A) - EMC Directive 2004/108EC replaced by 2014/30/EU - New command: SSR (refer to §4.16) - Tangential velocity (refer to §6.4.8) - Minor changes |
| Ver K | 06.12.16 | Updated version (with firmware from version 3.12A) - New parameter K600 (refer to §6.2.4) |
| Ver L | 27.11.17 | Updated version (with firmware from version 3.14A) - Minor changes |
| Ver M | 07.12.18 | Updated version (with firmware from version 3.15A) - New command: CHKDISTGRT (refer to §4.2.6) - Continuous traces (refer to §4.12.6) - Performance aspects (refer to §7.7) |
| Ver N | 29.05.19 | Updated version (with firmware from version 3.17A) - UtimET Light compatible with the PCI standard 2.2 |
| Ver O | 10.12.19 | Updated version (with firmware from version 3.18A) - New monitoring: M443 and M447 (refer to §5.3.2) - Minor changes (refer to modification strokes) |
| Ver P | 14.09.20 | Updated version (with firmware from version 3.19A) - New monitoring: M519 (refer to §4.14.3) - New monitoring: M448 (refer to §5.3.2.1) |
| Ver Q | 06.03.21 | Updated version (with firmware from version 3.20A) - Minor changes (refer to modification strokes) |
| Ver R | 10.09.21 | Updated version (with firmware from version 3.21A) - Minor changes (refer to modification strokes) |
| Ver S | 25.08.22 | Updated version (with firmware from version 3.21A) - Minor changes (refer to modification strokes) - New command: RSU (refer to §4.5) |
| Ver T | 13.03.23 | Updated version (with firmware from version 3.50A) - Minor changes (refer to modification strokes) |

Documentation concerning the UltimET Light family:

- | | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> • UltimET User's Manual • AccurET/ACCURET+ Operation & Software Manual • AccurET Modular48 Hardware Manual • AccurET Modular300 Hardware Manual • AccurET Modular400-600 Hardware Manual • AccurET VHP48 Hardware Manual • AccurET VHP100 Hardware Manual • ACCURET+ 100 Hardware Manual • ACCURET+ 300 Hardware Manual • ComET User's Manual • EDI User's Manual | UltimET Light principle & operation Setup, use & programming manual Specifications & electrical interfaces Specifications & electrical interfaces Specifications & electrical interfaces Specifications & electrical interfaces Specifications & electrical interfaces Specifications & electrical interfaces Specifications & electrical interfaces Specifications & electrical interfaces Commissioning software EDI principle & operation |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Introduction

This document concerns the ETEL motion controller family called 'UltimET Light' (also called motion controller or simply UltimET in this document) used with ETEL's AccurET and ACCURET+ position controllers. The purpose of this manual is to give details regarding the functioning, installation, tuning, functions and programming possibilities.



The information provided in this manual is valid for UltimET Light firmware version 3.50A or above. AccurET and ACCURET+ position controllers interfacing with this motion controller must have a firmware version 3.50A or above. The versions of EDI and ComET used with this product must be 4.50A or above to support all its functionalities.

Any further mention about AccurET is valid for AccurET Modular/VHP and ACCURET+ controllers.

Remark: The updates between two successive versions are highlighted with a modification stroke in the margin of the manual.

Safety

| | |
|--|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | The user must have read and understood this documentation before carrying out any operation on an UltimET Light. Please contact ETEL or authorized distributors in case of missing information or doubt regarding the installation procedures, safety or any other issue. |
| | ETEL S.A. disclaims all responsibility to possible industrial accidents and material damages if the procedures & safety instructions described in this manual are not followed (including the ones given in the manuals listed page 9). |

- Never use the UltimET Light in operating conditions and for purposes other than those described in this manual.
- A competent and trained technician must install and operate the UltimET Light, in accordance with all specific regulations of the respective country concerning both safety and EMC aspects.
- The customer must provide at all time the appropriate protections against electrical hazard and moving parts of the connected system. Operating the controller will make the motor move.
- The safety symbols placed on the UltimET Light or written in the manuals must be respected.
- If the UltimET Light is integrated into a machine, the manufacturer of this machine must establish that it fulfills the 2014/30/EU directive on EMC before operating the UltimET Light.

| | |
|--|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| | Signals a danger of electrical shock to the operator. Can be fatal for a person. |
| | Signals a danger for the UltimET Light. Can be destructive for the material. A danger for the operator can result from this. |
| | Indicates electrostatic discharges (ESD), dangerous for the UltimET Light. The components must be handled in an ESD protected environment, only. |

General operating conditions

The UltimET Light is designed to operate in a non-aggressive and clean environment, with a humidity rate ranging between 10% and 85%, an altitude < 2000m (6562 ft), and a temperature ranging between +15°C (59°F) and +40°C (104°F). The electronics must be in an enclosure respecting a pollution degree of 2 (refer to UL 508C and EN 61800-5-1 standards for more information). The UltimET Light is not designed or intended for use in the on-line control of air traffic, aircraft navigation and communications as well as critical components in life support systems or in the design, construction, explosive atmosphere, operation and maintenance of any nuclear facility.

For canadian markets the transient suppression test according to CSA C22.2 No. 14-05, clause 4.15.3, shall be conducted in the end use application.



The UltimET Light motion controllers have been successfully tested and evaluated to meet the UL 508C for US market.

This standard describes the fulfillment by design of minimum requirements for electrically operated power conversion equipment which is intended to eliminate the risk of fire, electrical shock, or injury to persons being caused by such equipment.

THIS PAGE IS INTENTIONALLY LEFT BLANK

Chapter A: Internal functioning & generalities

1. UltimET Light principle

UltimET Light is a very powerful master developed to control ETEL's AccurET position controllers family in multi-axis applications thanks to its real-time deterministic management. Two different models are available: one for multi-axis synchronization and one for complex interpolated trajectories (refer to [§2.4](#) for the ordering information).

The UltimET Light communicates with the PC through either a PCI, a PCI express (PCIe) or a TCP/IP connection. The PC can run under the following operating systems: Windows 7 (32-bit and 64-bit), Windows 8 (32-bit and 64-bit), Windows 10 (32-bit and 64-bit) and RTX (32-bit and 64-bit).

The UltimET Light links, in a daisy chain, the controllers by the communication bus called TransnET (ETEL proprietary protocol). The TransnET enables the UltimET Light to have an ultra-fast management of all position controllers. The UltimET Light allows the management of up to 63 axes.

The motion controller version with interpolation can execute interpolated trajectories on 2 groups of up to 4 interpolation axes (X, Y, Z and θ directions) with a very fast cycle time (100 μ s). Each interpolation axis may represent several controller axes if axes are in gantry level 1 mode. If the gantry is in level 2, only the gantry master axis is required. The trajectories available with the UltimET Light can be used with a simple language based on G-Code or complete PVT (position-Velocity-Time), PT (Position-Time) or ULM (Universal Linear Movement) modes with points refreshed every 100 μ s.

The UltimET Light can be programmed by an internal user sequence with multitasking capabilities as up to 3 sequence threads can run simultaneously.

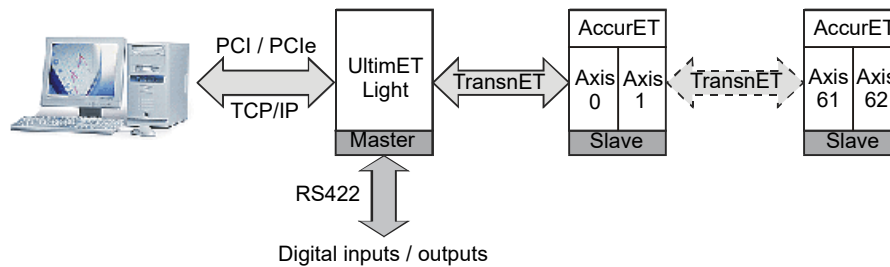
The user-friendly ComET software can be used to send commands, to write and download the user's sequences and to monitor the registers of the motion and position controllers. A software library including the UltimET Light and the AccurET functions is delivered to the user in order to use them with a user's PC-based program written in C, C++, C# and VB (NET framework version 4).

Different functions such as CAM possibility, fast internal inputs and outputs,... are described in this manual.

2. UltimET Light internal functioning

2.1 Communication with UltimET Light

The PC and UltimET Light are linked through PCI, PCIe or TCP/IP. All the position controllers are linked together to the UltimET Light (master) by the communication bus called TransnET (ETEL proprietary protocol). One of the roles of the master is to dispatch the orders received from the PC or (sent by itself) to the position controllers. Each axis has a personal number, and if several axes are chained, their number must be different from each others (from 0 to 62). The UltimET Light will always have the number 63. It is then possible to link up to 63 nodes.



The monitoring ***ML512** allows the user to know the number of nodes present on the TransnET communication bus. Refer to [§4.2](#) for more information about the monitorings.

| Monitoring | Comment |
|---------------|-----------------------------------------------------|
| *ML512 | Gives the mask of the nodes present on the TransnET |
| *M87 | Gives the UltimET axis number |

2.2 Several commands types

There are different types of commands in the UltimET Light:

- Normal command: general commands sets enabling the management of the different nodes, the procedure of the sequences (tests, loop...) or doing mathematical calculations (+, -, *, /...).
- Interpolation command: commands enabling the description of the interpolated movements done by the axes (circles, lines...).
- Urgent command: these commands enable a quick action on a node. They have specific channels which are reserved for them on the TransnET and are processed before the other types of commands. It is mainly security commands (immediate stop, stop with programmed deceleration, power off...). The urgent commands can be sent either from the 'Terminal' tool of ComET and in that case they are preceded by an exclamation mark or from the resident program of the PC.
- Monitoring command: these commands enable the user to obtain information about a node (position, speed, parameter value...). This type of command owns specific channels on the TransnET which allow it to work in parallel with the other types.

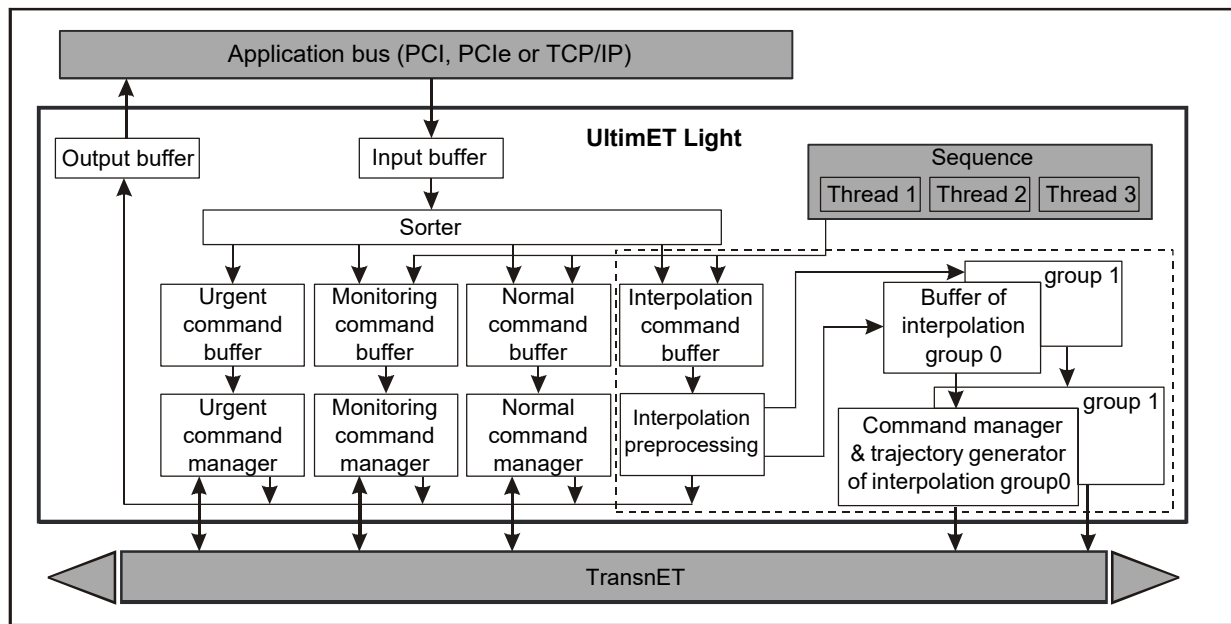
2.3 Block diagram

2.3.1 Communication

The UltimET Light has 2 communication ports:

- A PCI, PCIe or TCP/IP port. This port allows the card to communicate with a host PC and a client system to send commands to the UltimET Light and to receive information. To use this interface, ETEL has written functions libraries.
- and
- A TransnET port to communicate with ETEL's position controllers (AccurET). The TransnET is a numerical field bus with a data rate of 1Gbit/s. It allows the UltimET Light to communicate with ETEL's controllers, to send different commands, to transmit position dimensions and to obtain information from the controllers like the position of the axes.

High-level programs (with conditional instructions, loops, mathematical calculations, variables...) can be saved by the user in the UltimET Light, allowing an independent operation.



 : Only available with the interpolation version

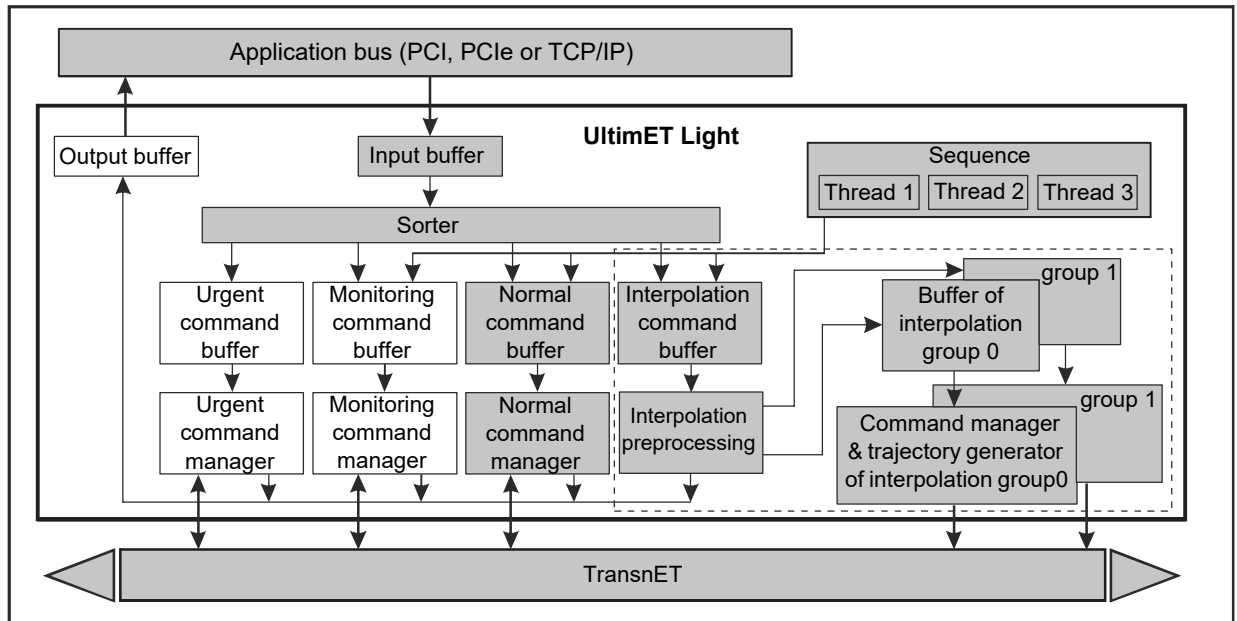
2.3.2 Several command sources

The commands sent to the different nodes (UltimET Light included) come from different sources. All the sources can be active without disrupting each other. The sources are either the sequences' thread of the user saved in the UltimET Light or the PC (via the PCI, PCIe or TCP/IP port of the UltimET Light).

The execution and the sending of the normal command type are done by the normal command manager (refer to the diagram below). Commands are processed in the order of arrival. If a command cannot be immediately executed (if a node is already occupied for example), the manager simply goes to the next one and will come back later to this one.

The TransnET channel number is not infinite and can be totally occupied. In this case, the commands pile up in the different buffers up to the saturation of the last one (input buffer) and to alert the PC. As soon as the TransnET will have free channel again, the commands will be sent.

The interpolation commands are all pre-processed, placed in the interpolation command buffer (1 per interpolation group) and executed by their interpolation command manager & trajectory calculator in the order of arrival.



--- : Only available with the interpolation version

2.3.3 Management of the interpolation commands

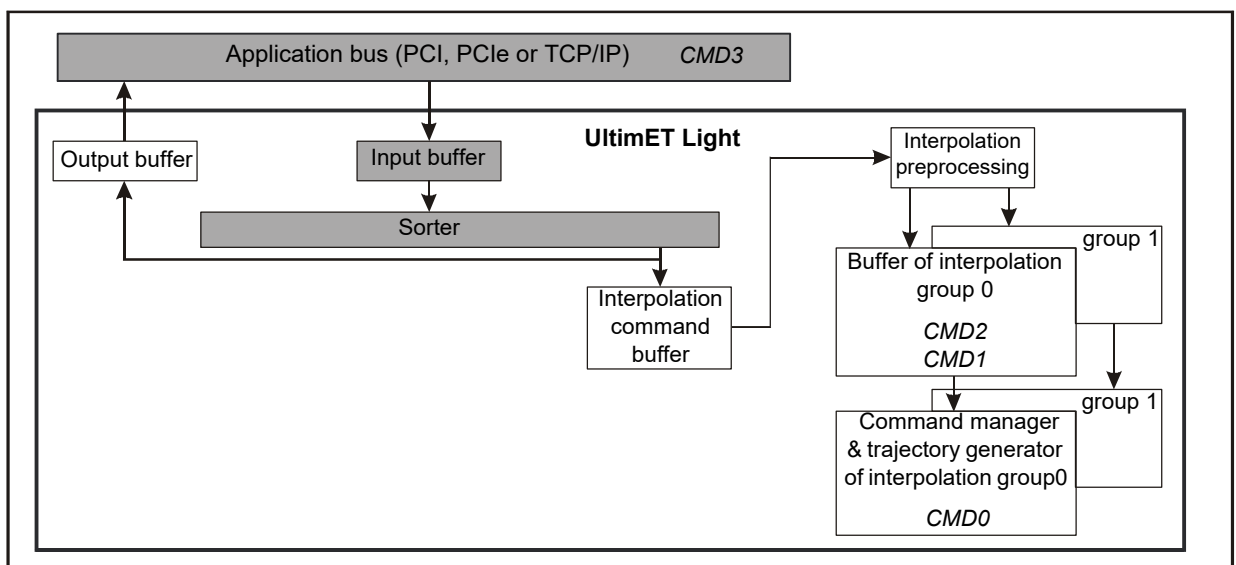
The interpolation commands are available only in the UltimET version with interpolation. The processing of these interpolation commands (*ILINE, *ICWR, *IMARK,...) is slightly different from the others. When a command arrives from the PCI, PCIe or TCP/IP port or a sequence, it is preprocessed and put in the corresponding interpolation buffer (1 per interpolation group) and is considered as finished (at the PCI, PCIe, TCP/IP or sequence thread level). It means that the sequence immediately goes to the next command or that the acknowledge is immediately returned to the PCI, PCIe or TCP/IP port.

After the interpolation buffer, the trajectory calculator takes the commands from the buffer, one after the other in the order of arrival to execute them. At this time the motors are going to move according to the movement defined by the command.

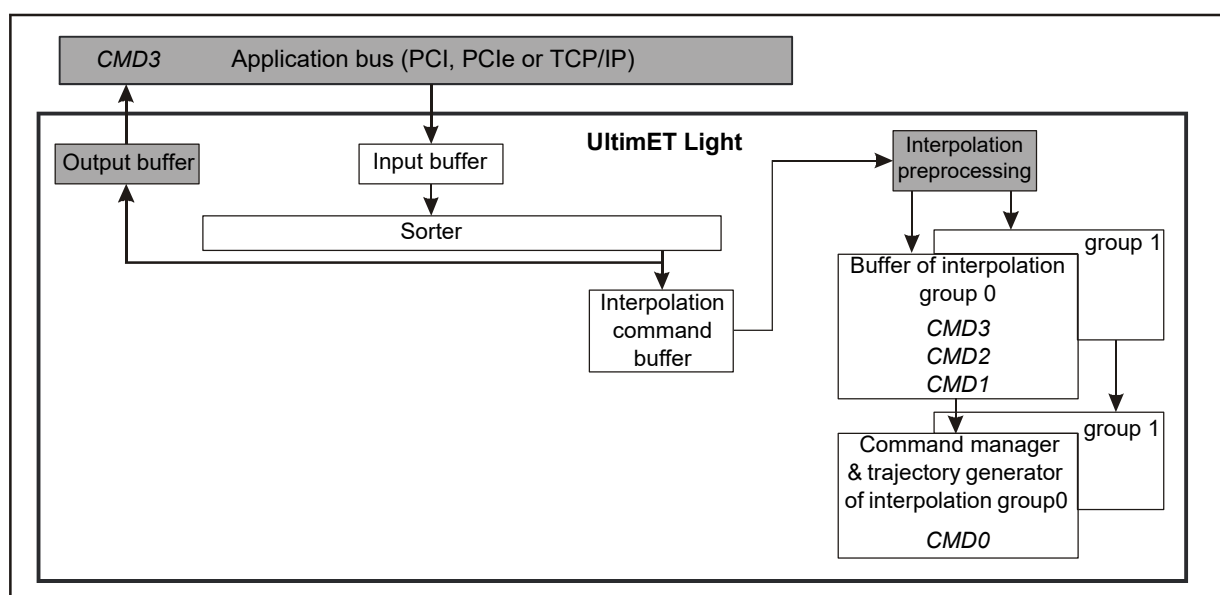
The time between the acknowledgment of the command and its real execution is equal to the sum of all the movements times defined by the commands already present in the buffer.

Example:

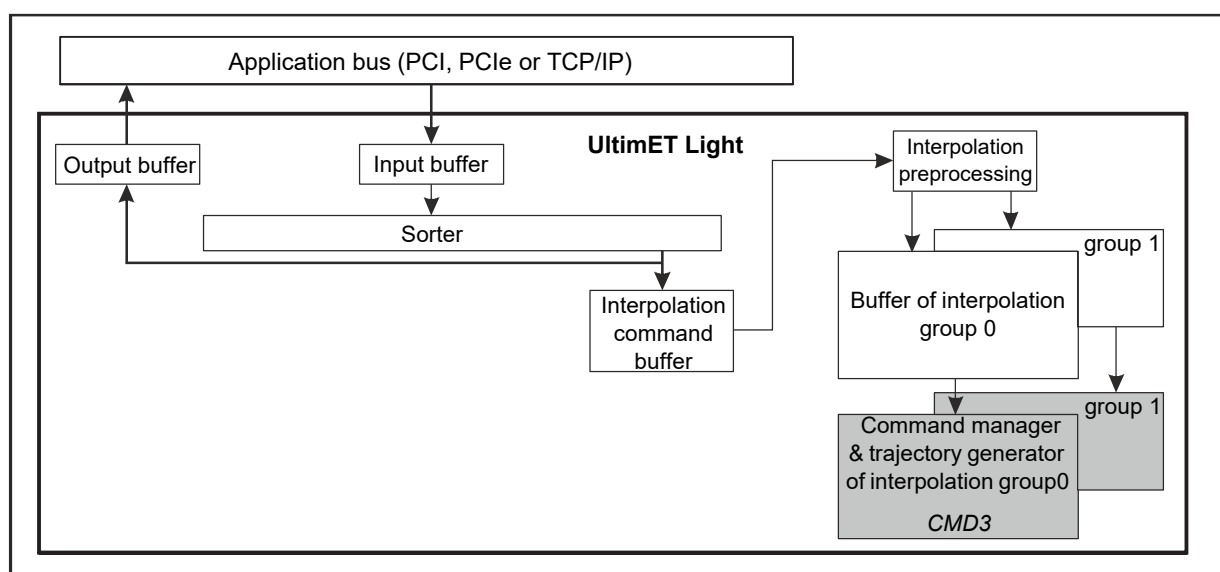
Here is the progression of an interpolation command. First, the CMD3 command, intended for the interpolation group 0 and coming from the PCI, PCIe or TCP/IP port, is preprocessed and goes to the interpolation buffer of the group 0, which already contains the CMD1 and CMD2 commands while the trajectory calculator executes CMD0.



Once CMD3 is in the buffer, it is immediately released from the PCI, PCIe or TCP/IP port. For the PCI, PCIe or TCP/IP port, CMD3 is finished.



When the trajectory calculator will have finished to execute CMD0, CMD1 and CMD2, it will deal with CMD3 and only at this time the movement defined by CMD3 will be executed by the motors.



A warning is raised when the buffer is nearly full. Bit#20 in *M166 and *K166 (refer to §4.13.6 for the warnings management) is set to indicate that one of the interpolation commands buffer is nearly full. The warning threshold can be set by the parameter *K513; it is expressed in percentage of the buffer size and represents the remaining size below which the warning is raised. The monitoring *M513 gives the equivalent in number of slots. The monitoring *M512 gives the actual number of free slots. By default, *K513 is 30% which means that if the buffer is (nearly) two thirds full, the warning is raised. Depths 0 and 1 correspond to interpolation groups 0 and 1.

| Parameter | Comment |
|-----------|--------------------------------------------------------------------------------------------------|
| *K513 | Percentage of the buffer capacity below which the 'interpolation buffer full' warning is raised. |

| Monitoring | Comment |
|------------|-------------------------------------------------------------------------------------|
| *M512 | Actual number of remaining free slots. |
| *M513 | Number of free slots below which the 'interpolation buffer full' warning is raised. |

Remark: To differentiate the commands going to the interpolation buffer from the other commands, refer to the UltimET Light commands list in §8.

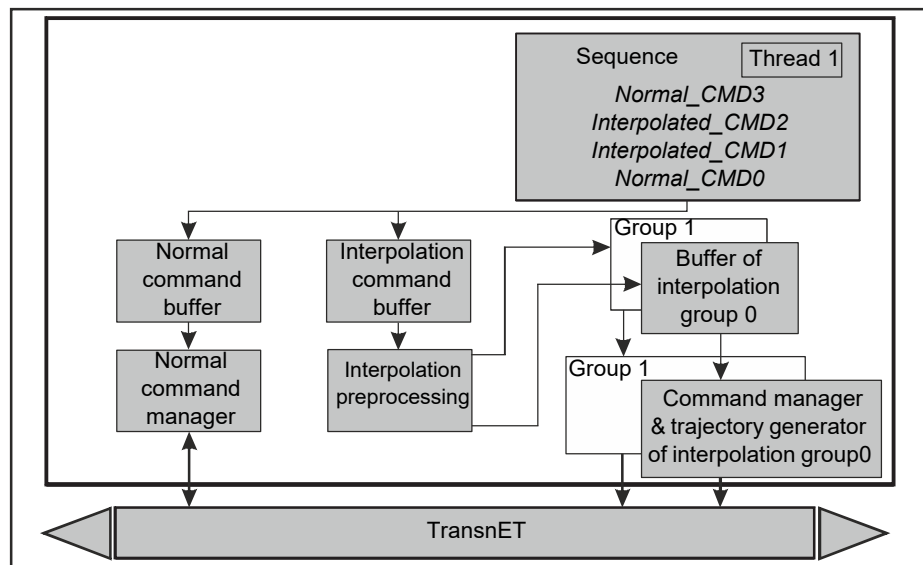
The monitoring ***M535** allows the user to know the number of commands present in the interpolation buffer.

| Monitoring | Comment |
|--------------|-------------------------------------------------------------------------------------------------------|
| *M535 | Number of commands present in the interpolation buffer (depth 0 for group 0 and depth 1 for group 1). |

Remark: To clear the buffer, refer to the ICLRB command (§6.4.5.2).

2.3.4 Management of a mix of normal and interpolated commands

We consider a series of commands coming from the thread 1 of the sequence. This sequence contains normal commands (PWR=1, *X1=10, DOUT.3=2,...) and interpolated commands (ILINE, IPVT,...), that are placed in a certain order.



The sequence processing dispatches commands on specific buffers, based on their type: normal or interpolated. This is done sequentially, line after line. Normal commands are stored in the normal command buffer. They are then processed by the normal command manager. Interpolated commands are stored in the interpolation command buffer. They are then pre-processed and sent to the specific interpolation group buffer. Pre-processing consists in pre-calculating data, for the trajectory generator to be efficient when processing a new interpolated segment.

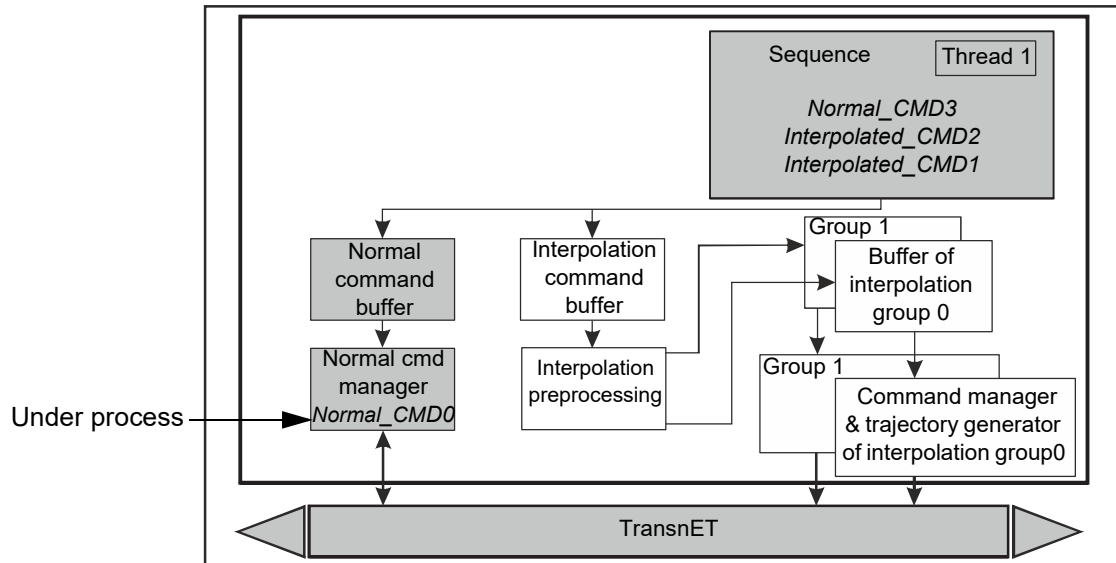
- Normal command manager: if the command concerns the UltimET (*X1=10,...), they are directly executed. If they concerned the axis (DOUT.1=2,...) they are sent trough the TransnET to the specific controller(s).
- Interpolation manager: the trajectory command is handled by the UltimET trajectory generator and target positions are sent to the controller(s) that are concerned by the interpolation group (as defined in *ISET command).

Then each command manager runs asynchronously. As soon as the last command is finished, the next command is processed.

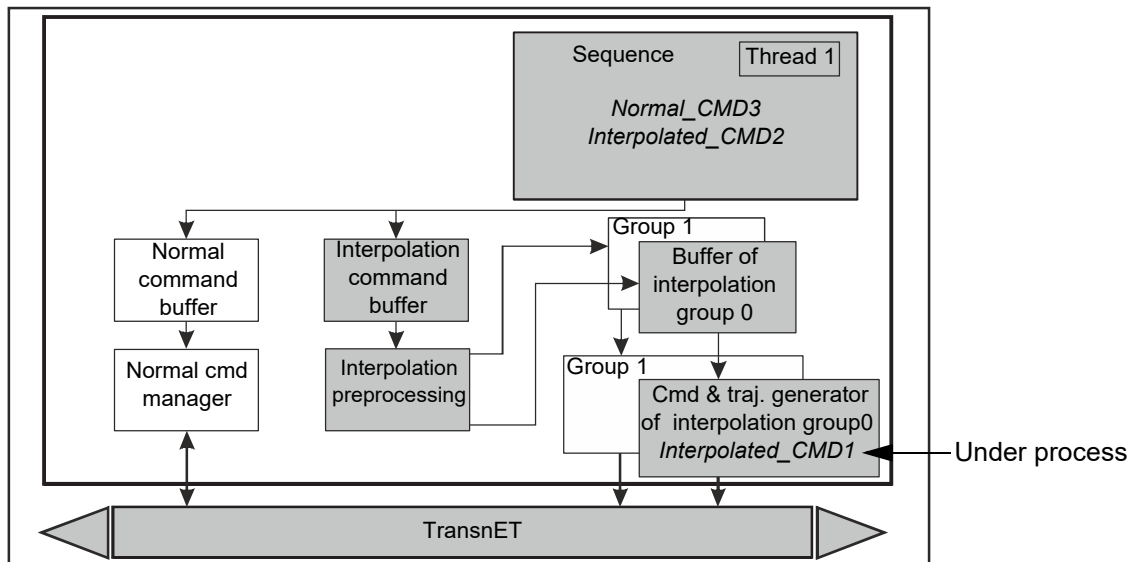
2.3.4.1 Command treatment order

Here is the order of the command treatment:

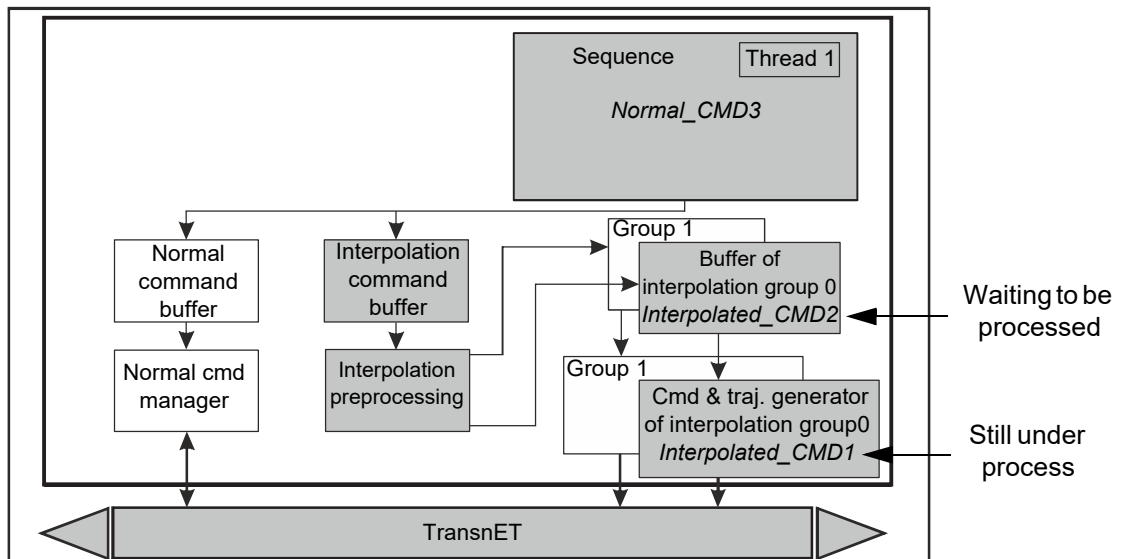
- 1) Normal CMD 0 is first stored in the normal command buffer. As soon as the normal command manager sees it, CMD 0 is processed.



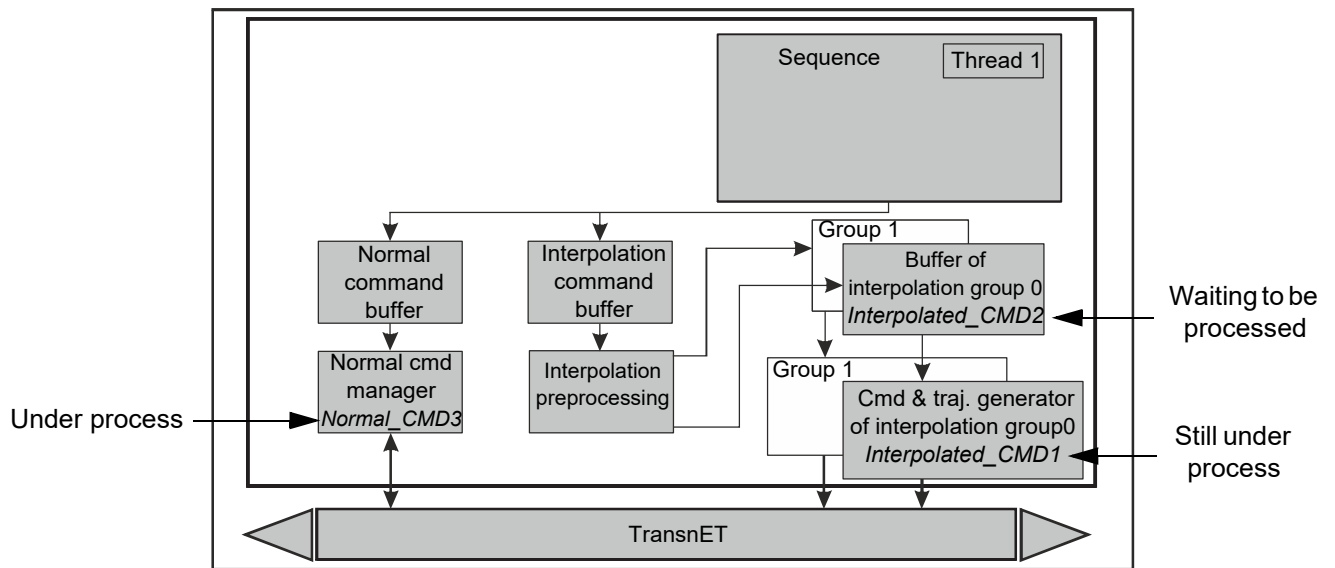
2) Interpolated CMD 1 is pre-processed and stored in the interpolation group 0 buffer. As soon as the command manager sees it, CMD 1 starts to be processed.



3) Interpolated CMD 2 is pre-processed and stored in the interpolation group 0 buffer. As the command manager is still processing CMD 1 (on-going movement), CMD 2 remains in the buffer.



4) Finally CMD 3 is transferred to normal command buffer. As soon as the normal command manager ends with CMD 0 (*X112=0, DOUT.1=2,...), CMD 3 will be under process.

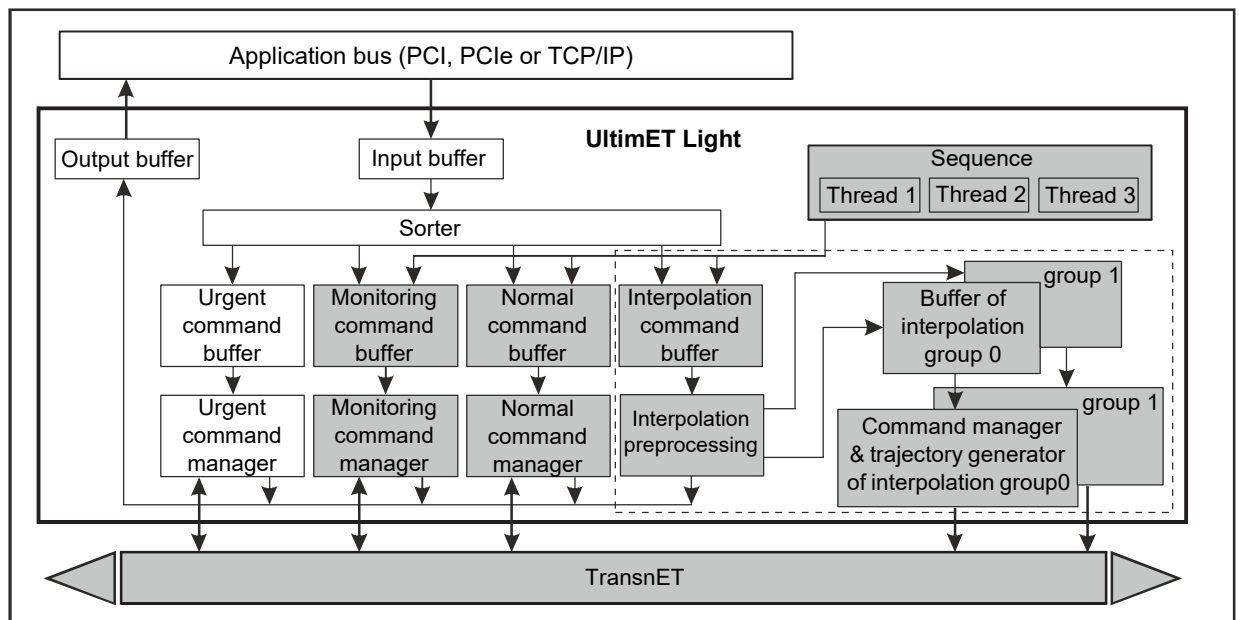


As soon as CMD1 will be treated (end of interpolated segment), CMD2 will be under process.

2.3.5 Multi-thread sequences

Several threads can run simultaneously on the board, each of them executing a different sub-sequence. Each thread has specific resources (variables and accumulator) enabling it to work regardless of the other threads. If necessary, common resources (global variables) allow the threads to communicate and to synchronize with each other. The system can also share the code between threads. The same routine (appearing only once in the sequences memory space) can be simultaneously executed by several threads (refer also to [§7.2.8](#)).

 : Only available with the interpolation version



2.4 Ordering information

Here is the ordering information describing the meaning of each digit present on the label of the UltimET Light:

| | | | | | | | | | | | | | | | | | | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | E | U | - | L | C | P | - | 0 | - | 0 | - | 1 | 0 | 0 | 0 | - | 0 | 1 |
| Family code: EU = Electronics UltimET motion controller | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | |
| Product type / Technology: L = Light | | | | | | | | | | | | | | | | | | |
| Control: C = With interpolation (synchronized & interpolated movement) G = Smart Gateway (synchronized movement only) | | | | | | | | | | | | | | | | | | |
| Format: A = AccurET optional board format (TCP/IP communication) P = PC based board format (PCI communication) E = PC based board format (PCI Express communication) | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | |
| Reserved | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | |
| Reserved | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | |
| External I/O connectivity: 0 = No external I/O available (discontinued) 1 = External I/O available (refer to §3.1.2.3 for more information) | | | | | | | | | | | | | | | | | | |
| Reserved | | | | | | | | | | | | | | | | | | |
| Standard: 0 = UL 508C compliant | | | | | | | | | | | | | | | | | | |
| Reserved | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | |
| Hardware revision: 00 = Standard product (discontinued except for TCP/IP format) 01 = Standard product. Hardware revision 2 | | | | | | | | | | | | | | | | | | |

Chapter B: Installation and setting

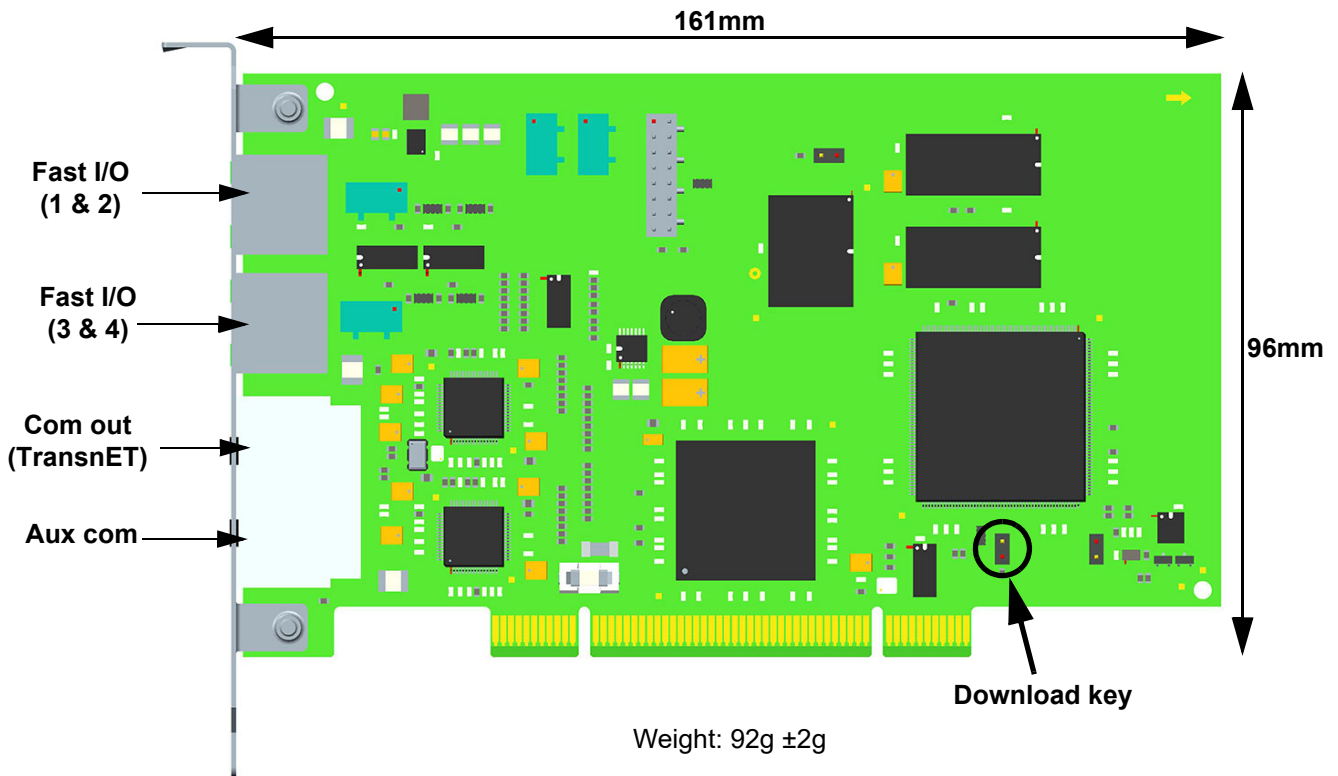
3. Initial installation

3.1 UltimET Light PCI

3.1.1 Hardware characteristics



The board must be handled in an ESD protected environment only.



| Characteristics | Value |
|----------------------------------------------------------------------|-------------|
| Input voltage | 3.3 VDC ±9% |
| Power input | 6 W |
| Littlefuse, NANO2, very fast-acting fuse for over-current protection | 125 VDC, 5A |

Remark: The UltimET Light is compatible with the PCI standard 2.2, 32x33Mhz. The UltimET Light only supports 3.3V signals.

3.1.2 Electrical interface

This chapter describes the pin assignment of the connectors.




The communication connectors must be insulated (no contact) from the power and mains lines. The inputs and outputs of these connectors are not galvanically insulated from the GND. The inputs and outputs must be connected to an Extra Low Voltage circuit only (SELV).




The board must be handled in an ESD protected environment only.

3.1.2.1 Fast I/O

The fast digital inputs / outputs follow the RS422 standard.

| RJ-45, 8 pins, female | | | |
|-----------------------------------------------------------------------------------|-------|---------|--------------------|
| FAST I/O (1 & 2) | Pin # | Signal | Function |
|  | 1 | DIN1 + | Digital input 1 + |
| | 2 | DIN1 - | Digital input 1 - |
| | 3 | DOUT1 + | Digital output 1 + |
| | 4 | DIN2 + | Digital input 2 + |
| | 5 | DIN2 - | Digital input 2 - |
| | 6 | DOUT1 - | Digital output 1 - |
| | 7 | DOUT2 + | Digital output 2 + |
| | 8 | DOUT2 - | Digital output 2 - |

| RJ-45, 8 pins, female | | | |
|-----------------------------------------------------------------------------------|-------|---------|--------------------|
| FAST I/O (3 & 4) | Pin # | Signal | Function |
|  | 1 | DIN3 + | Digital input 3 + |
| | 2 | DIN3 - | Digital input 3 - |
| | 3 | DOUT3 + | Digital output 3 + |
| | 4 | DIN4 + | Digital input 4 + |
| | 5 | DIN4 - | Digital input 4 - |
| | 6 | DOUT3 - | Digital output 3 - |
| | 7 | DOUT4 + | Digital output 4 + |
| | 8 | DOUT4 - | Digital output 4 - |

The commutation times of the above-mentioned inputs and outputs are as follows:

| | Status | Typical | Maximum | Unit |
|-------|--------|---------|---------|------|
| DOUTs | 0 => 1 | 11 | 16 | ns |
| | 1 => 0 | 11 | 16 | ns |
| DINs | 0 => 1 | 18 | 35 | ns |
| | 1 => 0 | 18 | 35 | ns |

Remark: The above-mentioned times takes only the hardware into account.
Refer to [§3.1.1](#) for the location of the connectors
Refer to [§4.10](#) for more information about the programming of the inputs and outputs.

3.1.2.2 Com out

The 'Com out' (**COM**munication **OUT**put) carries the TransnET communication bus used to communicate between the UltimET and the AccurETs. The inputs and outputs of the TransnET's signal are merged into the same cable. The cable coming out from this connector (Com out) must be connected to the 'Com in' (**COM**munication **IN**put) connector of the corresponding controller.

Remark: The TransnET cable must meet the following characteristics: 1:1 shielded cable, category 5E SFTP with 8 wires. The TransnET communication bus is ETEL's property.

3.1.2.3 Aux com

This connector allows the user to communicate with an external I/O Wago module with an Ethernet TCP/IP. The protocol used on this connector is the standard one of Ethernet. Refer to [§4.11](#) for more information. One UltimET can access one Wago coupler. The maximum distance between the UltimET and the Wago module is 100 m.

Remark: The cable must meet the following characteristics: shielded cable, category 5E SFTP with 8 wires.

3.1.2.4 Download key

In case of trouble during a firmware download (PC power malfunction, etc), the content of the flash may be corrupted and the firmware may not properly restart. In order to recover from this situation, a download key (jumper) must be used. Refer to §3.1.3.4 to know how to use it.

3.1.3 Software characteristics

3.1.3.1 UltimET driver installation

To install the UltimET Light PCI driver, please refer to the "UltimET PCI-PCIe.pdf" file in EDI or ComET documentation directory.

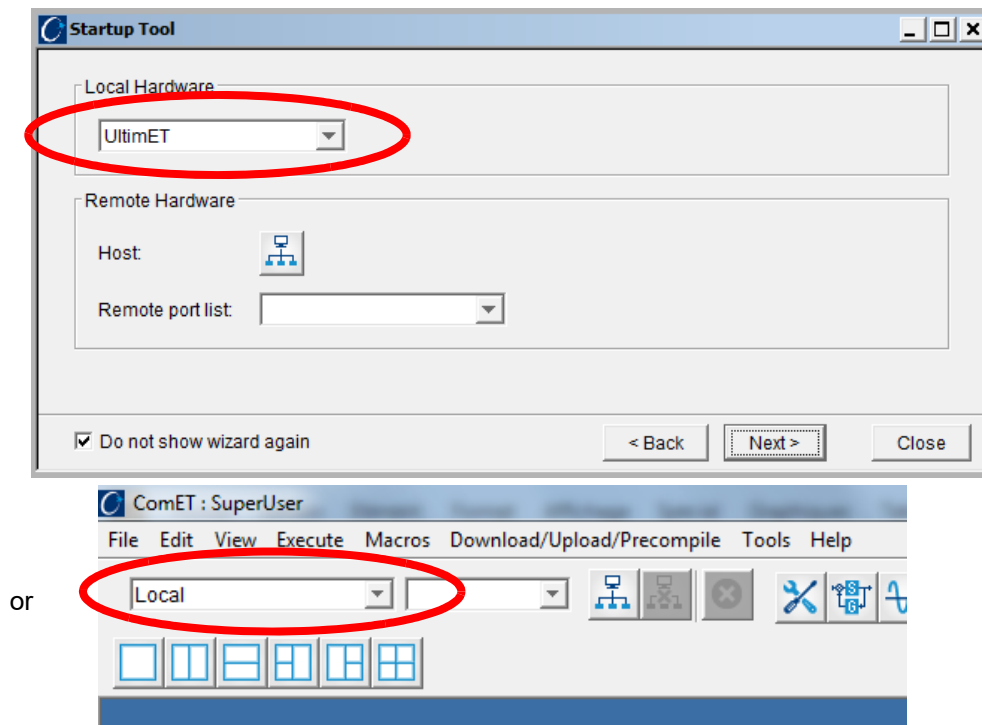
3.1.3.2 ComET software installation

The **ComET** software is a user-friendly interface developed by ETEL to set up and monitor the operation of the UltimET (and the associated controllers). The users who are not accustomed to work with it should read first the '**ComET User's Manual**'.

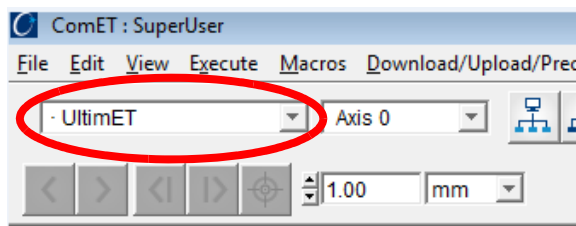
Remark: The minimum version of ComET compatible with the UltimET Light PCI is version 2.00A. The folder of installation as well as the backup of the files support **ONLY** standard characters and **NOT** the extended characters.


3.1.3.3 Establishment of the communication

The communication must be established prior to any operation otherwise the tools cannot be used. To do so, select 'UltimET' in the 'Startup tool' or 'local' menu ('UltimET PCI reset' = reset the UltimET Light's parameters).



When the communication is established, 'UltimET' appears in the top right corner:

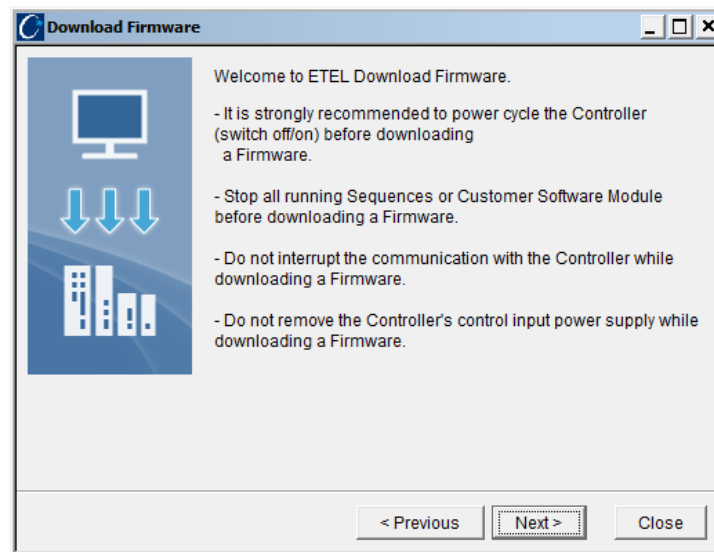


Remark: To stop the communication, click on the following button:  Refer to the '**ComET User's Manual**' for more information.

3.1.3.4 Firmware download

To download a new firmware, click on '**Controller**' in the menu bar and then on '**Download Firmware**'... From now on, follow step by step the windows given by the '**Download Wizard**' tool. It is required to stop all sequence threads before starting the firmware download process.

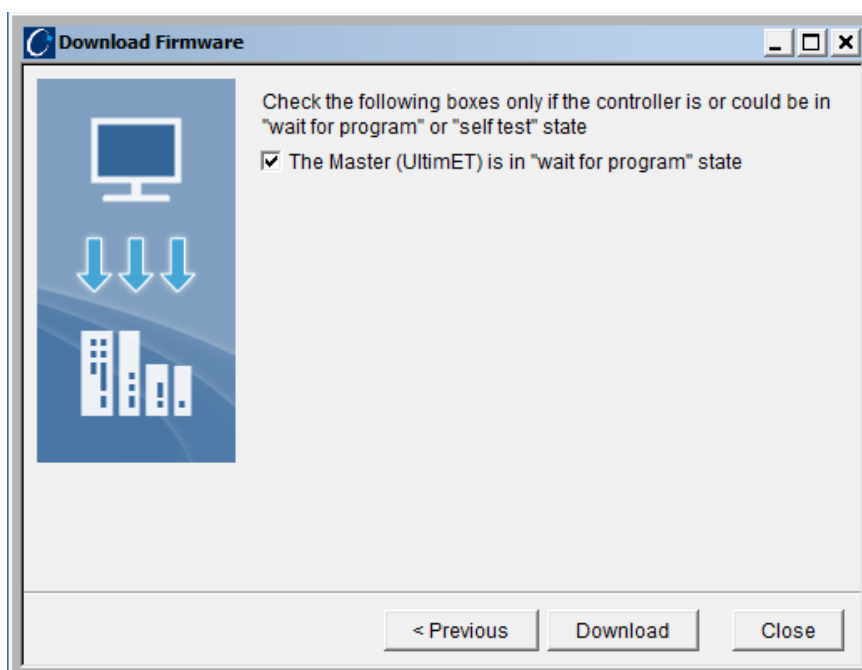
- Up to firmware 2.06A, when a new firmware is downloaded, the links with the sequence are lost and it is then not possible to execute it. It is then necessary to have a backup of the sequence in a file to download it again once the firmware download is done. If this backup has not been done before the firmware download and if the original sequence file is not found, it is necessary to download the previous firmware version to be able to upload the sequence in a file from the controller. However it will work only if the sequence source was successfully downloaded.
- When upgrading from a firmware $\leq 2.06A$ to a greater version, it is necessary to re-download the sequence (as mentioned above) otherwise the '**FORMAT SEQ ERR**' error (M64=460) will occur.
- For any upgrade with firmwares greater than 2.06A, it is not needed to re-download the sequence.



Remark: The name of the firmware file must have the **.far** extension.
After a firmware download, restarting the firmware without a complete power off of the PCI/PCle interface (computer restart) may raise the error "FPGA version is not the expected one" (*M64=4000). This is due to the fact that the firmware contained a new FPGA, and these can only be effectively taken into account after a PC power off. The user must shutdown and reboot the host PC.

If there is a problem during the download of the firmware, the following procedure must be applied:

- Switch off the PC and unplug the UltimET Light PCI (be ground connected while doing this)
- Plug a jumper on the download key position (refer to [§3.1.1](#)), put the board back into the PC and restart it
- Start ComET software and, without opening the communication with UltimET Light, execute the 'download firmware' tool (refer to the '**ComET User's Manual**' for more information)
- Select the appropriate UltimET Light firmware and follow the download procedure until the last panel, before pressing the 'download' button and select the option 'The UltimET Light is in 'wait for program' state' and click on 'download' button



- At the end of the download, switch off the PC, remove the UltimET Light PCI board, remove the jumper, put the board back into the PC and reboot the PC.

Remark: If after that, there is still a problem with the download of the firmware, please contact ETEL's technical support.

3.1.4 LEDs meaning

The LEDs present on the UltimET Light front panel have the following meanings:

| LED | LED number | Color | Meaning (if ON) |
|-----|------------|--------|----------------------------------------------|
| | 1 | Red | UltimET is in error if blinks at 1Hz |
| | 2 | Green | UltimET is working properly if blinks at 1Hz |
| | 3 | Red | User's LED managed by parameter *K595 |
| | 4 | Green | User's LED managed by parameter *K595 |
| | 5 | Yellow | TransnET link |
| | 6 | Green | TransnET is working properly |
| | 7 | Yellow | Ethernet I/O activity |
| | 8 | Green | Ethernet I/O link |

The parameter ***K595** allows the user to manage the user's LEDs number 3 and 4 of the UltimET Light PCI.

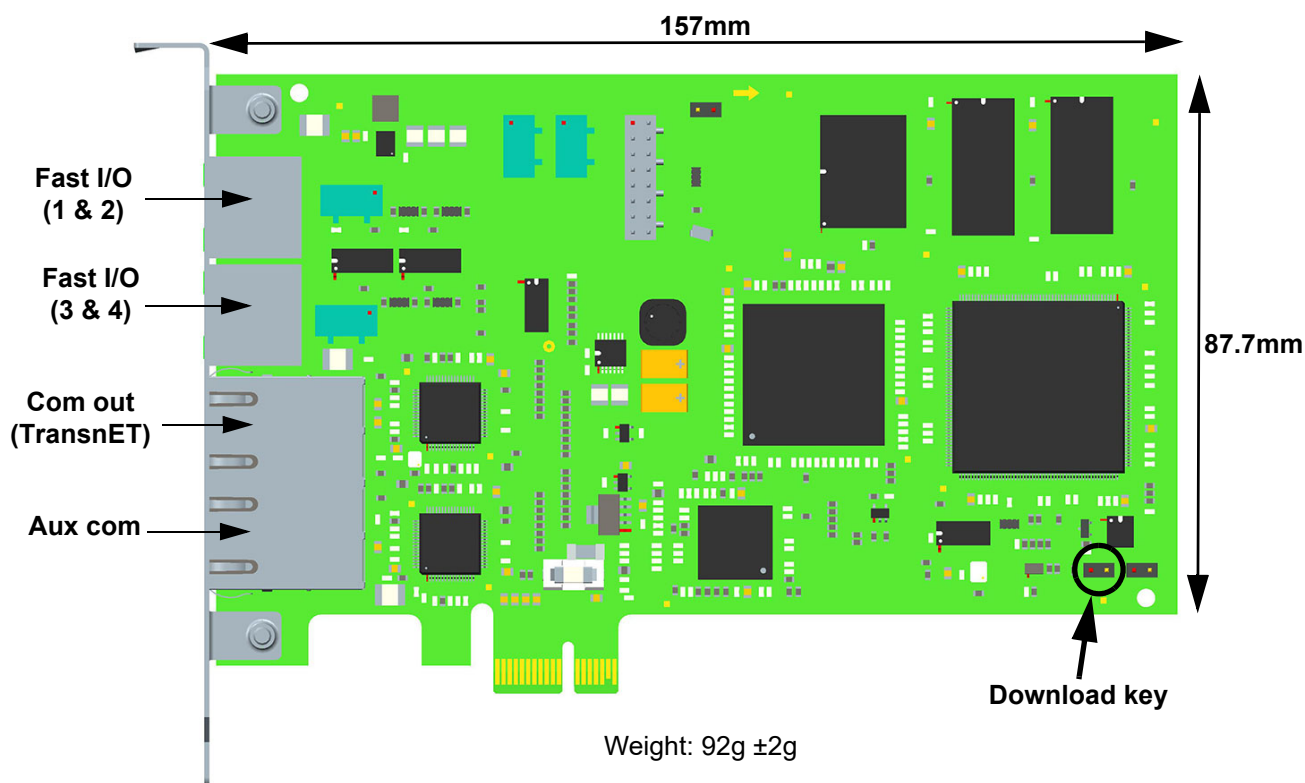
| Parameter | Comment |
|--------------|---------------------------------------------------------------------------------|
| *K595 | Control of the user's LED (1=on, 0=off, depth 0 = green led, depth 1 = red led) |

3.2 UltimET Light PCIe

3.2.1 Hardware characteristics



The board must be handled in an ESD protected environment only.



| Characteristics | Value |
|----------------------------------------------------------------------|-------------------|
| Input voltage | 3.3 VDC $\pm 9\%$ |
| Power input | 7 W |
| Littlefuse, NANO2, very fast-acting fuse for over-current protection | 125 VDC, 5A |

Remark: The UltimET Light is compatible with the PCIe, gen1, 2.5 Gbps.

3.2.2 Electrical interface

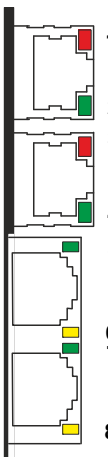
The electrical interface is totally identical to the one of the standard UltimET PCI. Please refer to [§3.1.2](#) for more information.

3.2.3 Software characteristics

The software characteristics are totally identical to the one of the standard UltimET PCI. Please refer to [§3.1.3](#) for more information.

3.2.4 LEDs meaning

The LEDs present on the UltimET Light front panel have the following meanings:

| LED | LED number | Color | Meaning (if ON) |
|-----------------------------------------------------------------------------------|------------|--------|----------------------------------------------|
|  | 1 | Red | UltimET is in error if blinks at 1Hz |
| | 2 | Green | UltimET is working properly if blinks at 1Hz |
| | 3 | Red | User's LED managed by parameter *K595 |
| | 4 | Green | User's LED managed by parameter *K595 |
| | 5 | Green | TransnET is working properly |
| | 6 | Yellow | TransnET link |
| | 7 | Green | Ethernet I/O link |
| | 8 | Yellow | Ethernet I/O activity |

The parameter ***K595** allows the user to manage the user's LEDs number 3 and 4 of the UltimET Light PCIe.

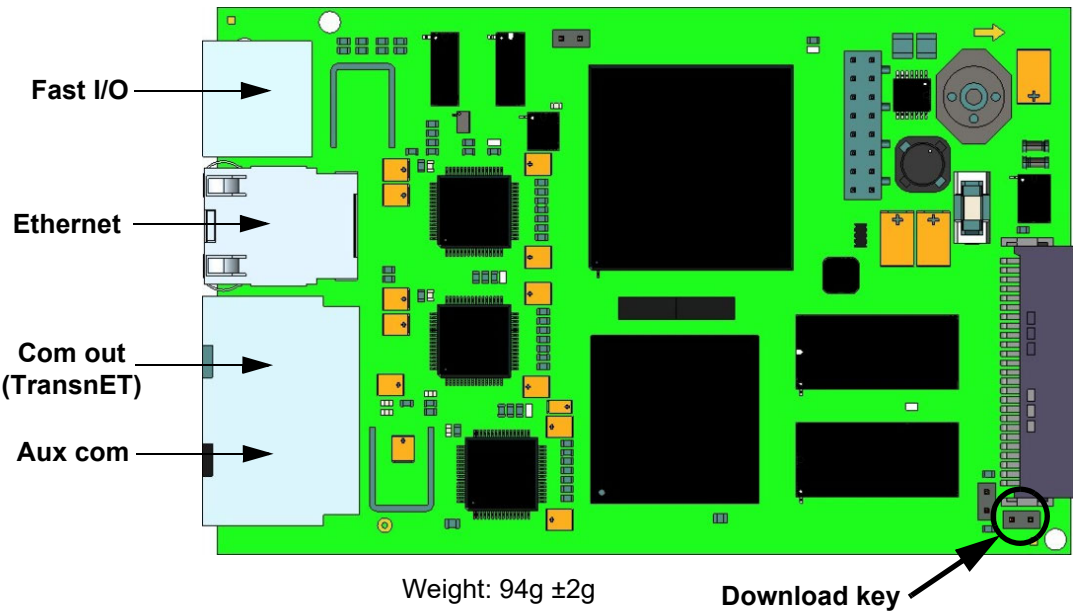
| Parameter | Comment |
|--------------|---------------------------------------------------------------------------------|
| *K595 | Control of the user's LED (1=on, 0=off, depth 0 = green led, depth 1 = red led) |

3.3 UltimET Light TCP/IP for AccurET optional board

3.3.1 Hardware characteristics





The board must be handled in an ESD protected environment only.



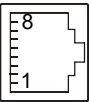
3.3.2 Electrical interface

This chapter describes the pin assignment of the connectors.

| | |
|-----------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | The communication connectors must be insulated (no contact) from the power and mains lines. The inputs and outputs of these connectors are not galvanically insulated from the GND. The inputs and outputs must be connected to an Extra Low Voltage circuit only (SELV). |
|  | The board must be handled in an ESD protected environment only. |

3.3.2.1 Fast I/O (connector U1)

The fast digital inputs / outputs follow the RS422 standard.

| RJ-45, 8 pins, female | | | |
|-----------------------------------------------------------------------------------|-------|---------|--------------------|
| FAST I/O | Pin # | Signal | Function |
|  | 1 | DIN1 + | Digital input 1 + |
| | 2 | DIN1 - | Digital input 1 - |
| | 3 | DOUT1 + | Digital output 1 + |
| | 4 | DIN2 + | Digital input 2 + |
| | 5 | DIN2 - | Digital input 2 - |
| | 6 | DOUT1 - | Digital output 1 - |
| | 7 | DOUT2 + | Digital output 2 + |
| | 8 | DOUT2 - | Digital output 2 - |

The commutation times of the above-mentioned inputs and outputs are as follows:

| | Status | Typical | Maximum | Unit |
|-------|--------|---------|---------|------|
| DOUTs | 0 => 1 | 11 | 16 | ns |
| | 1 => 0 | 11 | 16 | ns |
| DINs | 0 => 1 | 18 | 35 | ns |
| | 1 => 0 | 18 | 35 | ns |

Remark: The above-mentioned times takes only the hardware into account.
Refer to [§3.3.1](#) for the location of the connectors
Refer to [§4.10](#) for more information about the programming of the inputs and outputs.

3.3.2.2 Ethernet (connector U2)

This connector allows the user to establish the TCP/IP communication between the UltimET Light and the PC. The inputs and outputs of the TransnET's signal are merged in the same cable. The protocol used on this connector is the standard one of Ethernet.

Remark: The Ethernet cable must meet the following characteristics: 1:1 shielded cable, category 5E SFTP with 8 wires.

3.3.2.3 Com out (connector U3)

The 'Com out' (**COM**munication **OUT**put) carries the TransnET communication bus used to communicate between the motion controller and the controllers. The inputs and outputs of the TransnET's signal are merged into the same cable. The cable coming out from this connector (Com out) must be connected to the 'Com in' (**COM**munication **IN**put) connector of the corresponding controller.

Remark: The TransnET cable must meet the following characteristics: 1:1 shielded cable, category 5E SFTP with 8 wires. The TransnET communication bus is ETEL's property.

3.3.2.4 Aux com (connector U4)

This connector allows the user to communicate with an external I/O module with an Ethernet TCP/IP. The protocol used on this connector is the standard one of Ethernet. Refer to [§4.11](#) for more information. One UltimET can access one Wago coupler. The maximum distance between the UltimET and the Wago module is 100m.

Remark: The cable must meet the following characteristics: shielded cable, category 5E SFTP with 8 wires.

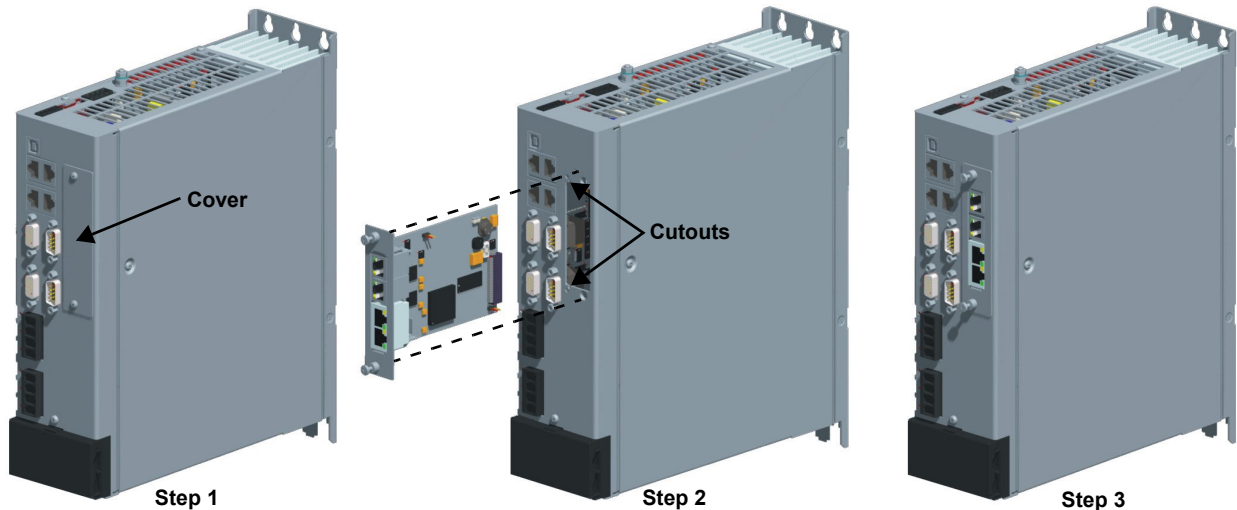
3.3.2.5 Download key

In case of trouble during a firmware download (PC power malfunction, etc), the content of the flash may be corrupted and the firmware may not properly restart. In order to recover from this situation, a download key (jumper) must be used. Refer to [§3.3.4.3](#) to know how to use it.

3.3.3 Hardware installation

To install the UltimET Light motion controller inside the position controller, the user must use the following procedure:

- Work in an ESD protected environment, ground connected yourself.
- Turn off all the power supplies (main and control) and wait 10 minutes to allow the internal DC bus capacitors to discharge.
- Unscrew the two screws fastening the cover of the optional board area on the front panel of the controller (step 1).
- Slide carefully the UltimET Light motion controller inside the controller by putting the PCB in the two cutouts (step 2).
- Push the board until the connection with the internal back panel connector is done.
- Screw the two screws present on the front panel of the optional board (step 3)



Remark: To remove the UltimET Light from the controller, first turn off all the power supplies (main and control) and wait 10 minutes to allow the internal DC bus capacitors to discharge and then follow in the reverse order the opposite actions of the above-mentioned steps.
Refer to the ordering information ([§2.4](#)) to know which controller can accept an optional board.

3.3.4 Software characteristics

3.3.4.1 ComET software installation

The **ComET** software is a user-friendly interface developed by ETEL to set up and monitor the operation of the UltimET (and the associated controllers). The users who are not accustomed to work with it should read first the '**ComET User's Manual**'.

Remark: The minimum version of ComET compatible with the UltimET Light TCP/IP is version 2.02A. The folder of installation as well as the backup of the files support **ONLY** standard characters and **NOT** the extended characters.

3.3.4.2 Establishment of the communication

Two kind of information are requested to communicate with UltimET Light TCP/IP:


- The IP address of the board,
- The port used to access to the UltimET Light. 3 TCP/IP ports are available:
 - Port 1129 (UltimET Light TPC-IP 1, in ComET)

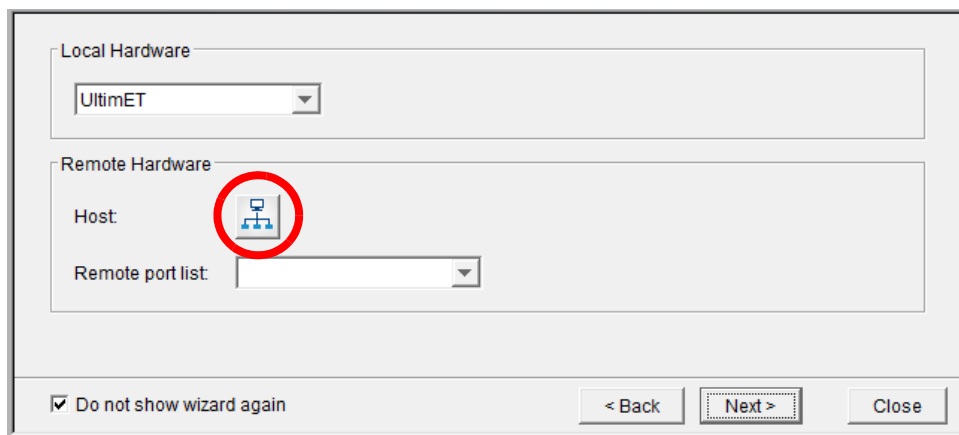
- Port 1128 (UltimET Light TPC-IP 2, in ComET)
- Port 1127 (UltimET Light TPC-IP 3, in ComET)

The communication with one port is limited to one connection. As an example, the user may connect its PC-based application through port 1127 and execute commissioning actions through port 1128. The communication must be established prior to any operation otherwise the tools cannot be used. The connection is the first 'next suggested action' proposed by ComET. This suggestion is highlighted in dark blue on the main screen.

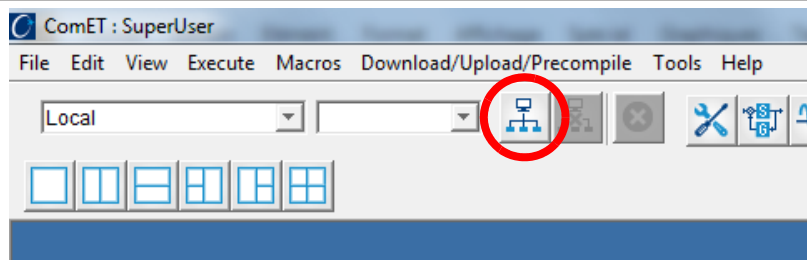
First of all, the user must establish the communication with the UltimET Light (and the controller(s)). To do so:

1° Double-click on the shortcut of your ComET.

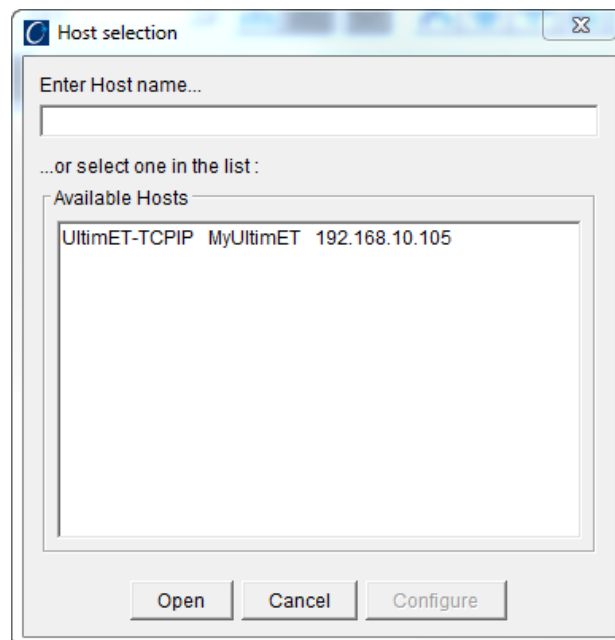
2° Click on the 'Remote hardware' icon  in order to find the UltimET Light connected to the network.



or



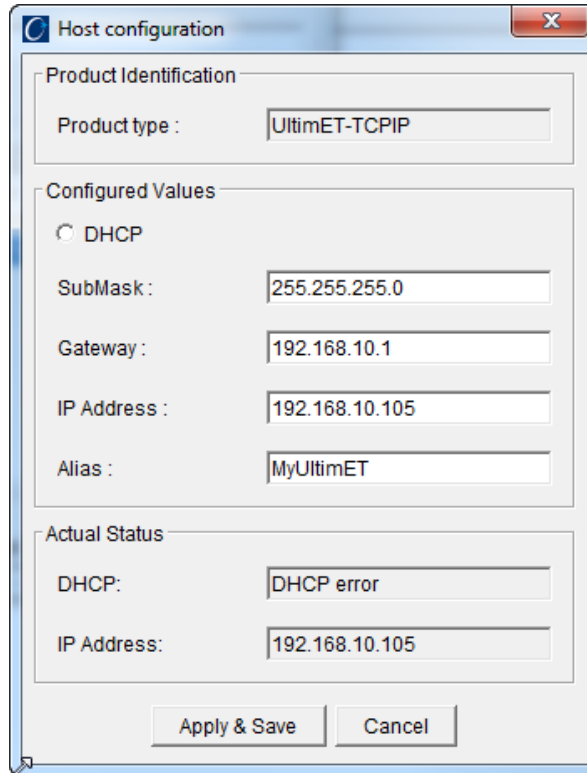
3° The **Host selection** window appears showing the UltimET Light connected to the network with their corresponding IP address.



The user can now select the wanted UltimET Light TCP/IP to communicate with (only one is present in the above example). The automatic detection of the UltimET Light TCP/IP board(s) is only valid when the UltimET Light TCP/IP is in the same sub-network as the PC running ComET.

Remark: By default the UltimET Light TCP/IP is configured with the IP address 172.22.10.102 and without DHCP.

4°a Once selected, the user can click on 'Configure' to change the network parameters of the UltimET Light TCP/IP. The following window appears:

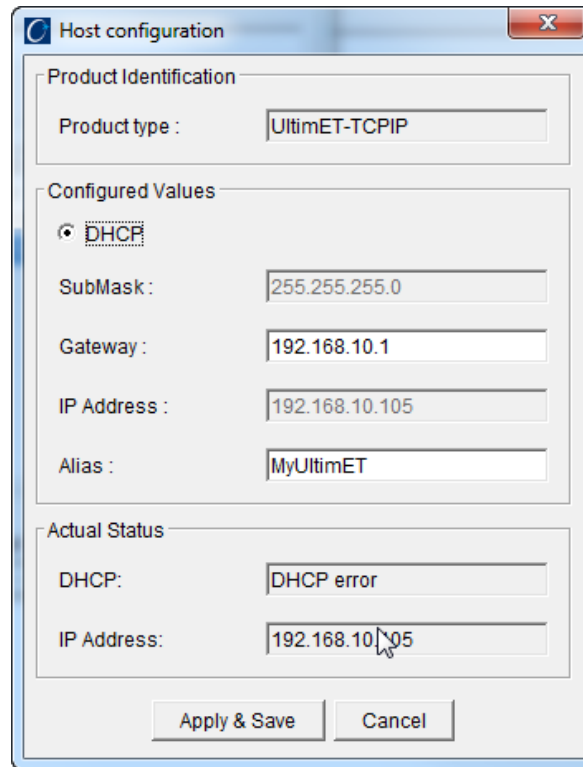


The image shows a 'Host configuration' dialog box with three main sections: 'Product Identification', 'Configured Values', and 'Actual Status'. In 'Product Identification', 'Product type' is set to 'UltimET-TCP/IP'. In 'Configured Values', the 'DHCP' radio button is selected, and fields for 'SubMask' (255.255.255.0), 'Gateway' (192.168.10.1), 'IP Address' (192.168.10.105), and 'Alias' (MyUltimET) are filled. In 'Actual Status', 'DHCP' is 'DHCP error' and 'IP Address' is '192.168.10.105'. At the bottom are 'Apply & Save' and 'Cancel' buttons.

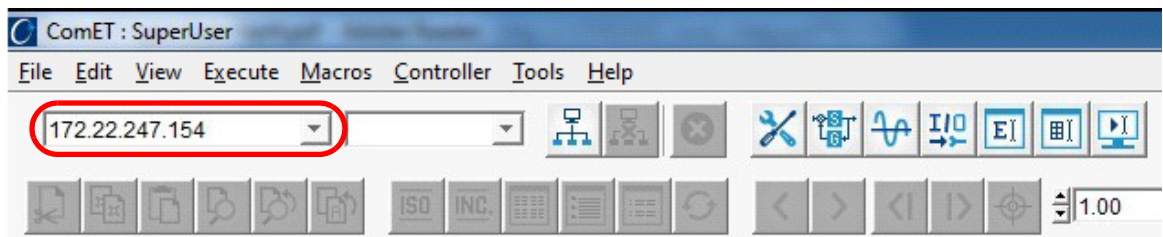
| Section | Field | Value |
|------------------------|---------------------------------------|----------------|
| Product Identification | Product type : | UltimET-TCP/IP |
| Configured Values | <input checked="" type="radio"/> DHCP | |
| | SubMask : | 255.255.255.0 |
| | Gateway : | 192.168.10.1 |
| | IP Address : | 192.168.10.105 |
| | Alias : | MyUltimET |
| Actual Status | DHCP: | DHCP error |
| | IP Address: | 192.168.10.105 |

The 'Gateway' field allows the user to enter the gateway IP address to be connected to another sub-network. If a change has been done, the user must click on 'Apply & Save' to save the new parameters and then the UltimET Light TCP/IP board must be switched off (and on) to take the new parameters into account.

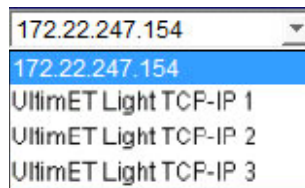
4°b With the DHCP server option, the UltimET Light will ask the DHCP server for an IP address at each power on. It works only if a DHCP server is present on the network. When the UltimET Light TCP/IP is waiting for an IP address from the DHCP server, the status LEDs have the following behavior: the red LED remains ON and the green one is blinking at 1Hz. After 60 seconds, if no DHCP server answers, the UltimET enters in error and use the fixed IP address saved in flash (or the default one), it is then possible (it is not possible during the DHCP process) to change the network configuration with ComET. If a change has been done, the user must click on 'Apply & Save' to save the new parameters:



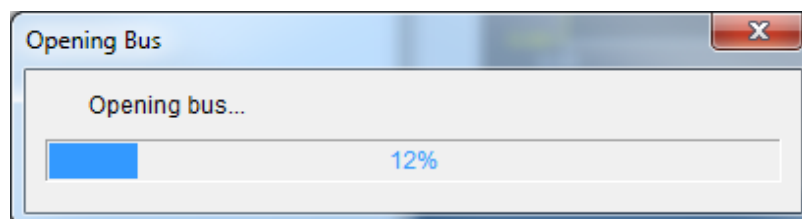
5° Now, click on 'Open' in the 'Host selection' window. The following window appears with the IP address corresponding to the selected UltimET Light TCP/IP board.



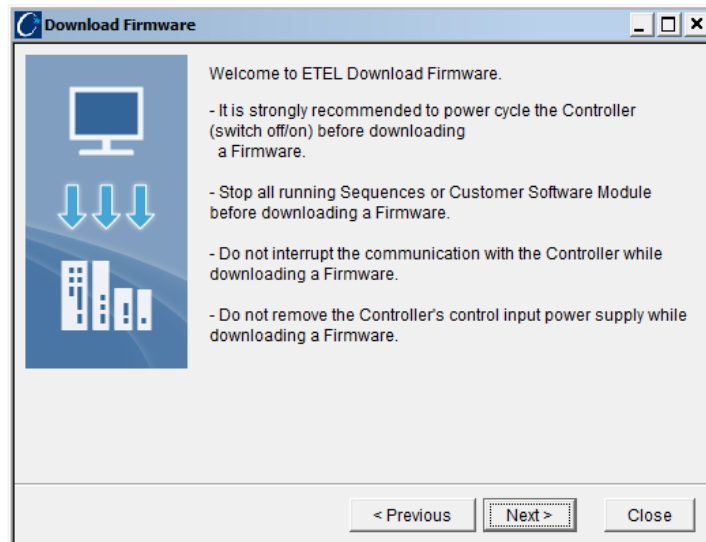
6° Now, click on the pull-down menu (beside the IP address) and choose one of the three port numbers to communicate with:



Once selected the communication with the UltimET Light TCP/IP board is establish:



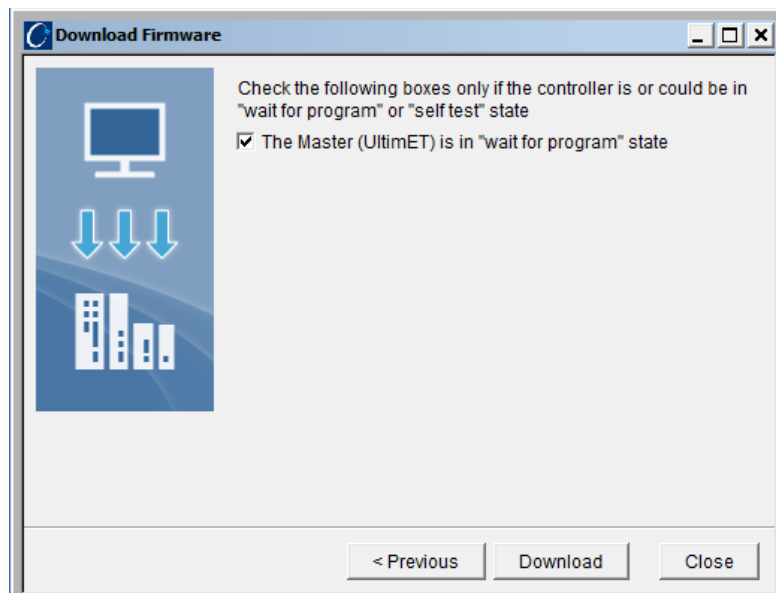
- For any upgrade with firmwares greater than 2.06A, it is not needed to re-download the sequence.



Remark: The name of the firmware file must have the **.far** extension.

If there is a problem during the download of the firmware, the following procedure must be applied:

- Switch off the controller(s) and unplug the UltimET Light TCP/IP by applying the procedure mentioned in [§3.3.3](#) in the reverse order (be ground connected while doing this)
- Plug a jumper on the download key position (refer to [§3.3.1](#)), put the board back into the controller and restart it
- Start ComET software and, without opening the communication with UltimET Light, execute the 'download firmware' tool (refer to the '**ComET User's Manual**' for more information)
- Select the appropriate UltimET Light firmware and follow the download procedure until the last panel, before pressing the 'download' button and select the option 'The UltimET Light is in 'wait for program' state' and click on 'download' button



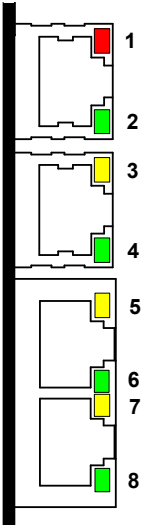
- At the end of the download, switch off the controller(s), remove the UltimET Light TCP/IP board, remove the jumper, put the board back into the controller and switch it on

If you meet problems like timeouts while downloading the firmware via TCP/IP, it should be ensured that the UltimET Light TCP/IP is not connected to the company's LAN. Instead, it should be considered putting it into a sub-network or better establishing a point to point connection between PC and UltimET Light TCP/IP.

Remark: If after that, there is still a problem with the download of the firmware, please contact ETEL's technical support.

3.3.5 LEDs meaning

The LEDs present on the UltimET Light front panel have the following meanings:

| LED | LED number | Color | Meaning (if ON) |
|-----------------------------------------------------------------------------------|------------|-----------------|----------------------------------------------|
|  | 1 | Red | UltimET is in error if blinks at 1Hz |
| | 2 | Green | UltimET is working properly if blinks at 1Hz |
| | 3 | Green or yellow | Ethernet activity |
| | 4 | Green | Ethernet link |
| | 5 | Yellow | TransnET link |
| | 6 | Green | TransnET is working properly |
| | 7 | Yellow | Ethernet I/O activity |
| | 8 | Green | Ethernet I/O link |

Chapter C: UltimET Light commands

4. Commands & registers syntax

4.1 Commands

The «*» sign in front of a command enables the identification of the UltimET Light as the addressee of the command. This sign has the same meaning as 'axis' in the AccurET position controller syntax. The combination of both notations is possible to refer to several addressees (refer to the example below).

Syntax:

***<cmd_name>[.<axis>] [=<P1> [,<P2>]...]**

Fields put in 'square brackets' (like: [=<P1>]) are **optional**. All commands do not use them.

* The command is sent to the UltimET.

<cmd_name> Command name. All UltimET's commands have three letters or more.

.<axis> Axis or group of axis number which have to execute the command.

Possible values:

- Integer from 0 to 62 according to the number of axis if the command refers to one single axis.
- Symbol ! if the command refers to all linked axes.
- Some selected axes numbers between commas.

[=<P1> [,<P2>]...] The command can have zero, one, two or more parameters (note: the = link sign is needed only if at least one [<Px>] field is present).

Possible values:

- Immediate value in integer 32 bits or 64 bits, or in float 32 bits.
- Value contained in any register, at any depth. [Px] syntax is similar to a command syntax: *<register>[:<depth>].<axis>. Refer to [§4.2.1.1](#) for more information.

Note: If the command and its (or some) parameters refer to different axes, the UltimET will first fetch the actual parameter values and then substitute them in the command sent to the axis.

Example:

```
*RST.1;           // Command without parameter, sent to UltimET and axis 1.
*SAV.1 = 1;       // Command with one parameter (P1=1), sent to UltimET and axis 1.
*SAV.(1,2,8) = 1; // Command with one parameter (P1=1), sent to UltimET and the
                  // axes 1, 2 and 8.
*ILT.1 = 10,20;   // Command with two parameters (P1=10 and P2=20), sent to UltimET
                  // and axis 1.
```

Remark: If at least one parameter (P1,...) of a command is wrong (bad format for example), an error occurs.

4.2 Registers

4.2.1 Registers group

The **registers** are accessible to the user, they store all the UltimET Light's internal values. Each register has an identification number preceded by one or two letters corresponding to its group. Some registers belong only to the UltimET Light and some others are common to both UltimET Light and AccurET controller.

There are 6 main types of registers, 3 are basic (often or always used) and 3 are advanced (for specific applications only).

4.2.1.1 Basic registers

- K, for parameters** They define the configuration of UltimET and the setting of its features. Refer to §9. for the complete lists of K parameters. The parameters can be either common with the AccurET controller or dedicated to the UltimET Light.
- M, for monitorings** They are exclusively used to monitor the UltimET Light's internal values such as mask of the present nodes, interpolation active state, etc. They can only be read, and no value can be assigned. Refer to §10. for the complete lists of M monitorings.
- X, for user variables** They are variables that the user may freely use for programming. Each user variable function may be defined in a program, according to the user's needs. Values can be stored in variables and read at any time.

Depending on the type of the associated value, a letter must be added to the one of the corresponding register: 'L' for 'Long' (integer 64 bits), 'F' for 'Float' (float 32 bits) and 'D' for 'Double' (float 64 bits). This rule can be applied to all registers. The registers name becomes as follows:

| Register | Integer | Integer 64 bits | Float | Double |
|----------|---------|-----------------|-------|--------|
| K | K | KL | KF | KD |
| M | M | ML | MF | MD |
| X | X | XL | XF | XD |

Registers with a same number but of different type are not linked. They are independent and have their own memory address. The maximum number of the different registers varies according to the type of associated value:

| | Register | Integer | Integer 64 bits | Float | Double |
|---------------------------------------------|----------|---------------------|----------------------|----------------------|----------------------|
| Common to AccurET position controller | K | $0 \leq K \leq 511$ | $0 \leq KL \leq 511$ | $0 \leq KF \leq 511$ | $0 \leq KD \leq 511$ |
| | M | $0 \leq M \leq 511$ | $0 \leq ML \leq 511$ | $0 \leq MF \leq 511$ | $0 \leq MD \leq 511$ |
| | X | $0 \leq X \leq 511$ | $0 \leq XL \leq 255$ | $0 \leq XF \leq 511$ | $0 \leq XD \leq 255$ |

| | Register | Integer | Integer 64 bits | Float | Double |
|------------------------------|----------|-----------------------|------------------------|------------------------|------------------------|
| Specific to UltimET Light | K | $512 \leq K \leq 767$ | $512 \leq KL \leq 767$ | $512 \leq KF \leq 767$ | $512 \leq KD \leq 767$ |
| | M | $512 \leq M \leq 767$ | $512 \leq ML \leq 767$ | $512 \leq MF \leq 575$ | $512 \leq MD \leq 575$ |

The format of the corresponding values will be for example (as written in the 'Terminal' of ComET):

| ISO | Integer | Long | Float | Double |
|-------|---------|---------|-------|--------|
| 123.0 | 123 | 123L | 123F | 123D |
| 1.2 | 1 | 0x12ABL | 1.2F | 1.2D |

Remark: An **alias** is a more user-friendly term, like DOUT (for digital output),..., representing a parameter K or a monitoring M. Their syntax is identical to the one of the corresponding register. Refer to §9. and §10. to know the list of the alias.

4.2.1.2 Advanced registers

- L, for LKT** Also called **look-up table registers**. They are general purpose registers for temporary logs or measurements. They are only available in 32-bit integer and therefore, they will always be addressed with the L register. The first L register is L0 and the last one L1999. They are 16 depths available.
- T, for trace** They allow the acquisition of registers X, K, M and L of the UltimET Light according to the time. They are used by the 'Scope' of ComET ($0 \leq T \leq 16383$). According to the format of the register, the trace will correspond to T, TF, TL or TD.
- Y** Also called **advanced user registers**, they allow the change of value at any time in a PVT (Position-Velocity-Time) movement. As this register can only be associated to an 32-bit integer value type, it will always be written Y register ($0 \leq Y \leq 255$ on 4 depths).

4.2.2 Register value reading

Here is the syntax allowing the user to know the value in a register.

Syntax:

***<register> [:<depth>]**

Fields put in 'square brackets' (like: [:<depth>]) are **optional**. They are not always used.

<register> Defines the register used. It must include the **type** of register as well as its corresponding **number**:

Possible types:

- K, M, X, L, T, Y

Possible numbers:

- Integer (refer to [§4.2.1.1](#))

[:<depth>] Some registers may contain different values simultaneously. Each value is stored at a different **depth**. If no depth is defined, depth 0 is automatically used by default.

Possible values:

- Integer from 0 to 7 if the register type is **K**
- Integer from 0 to 3 if the register type is **X**
- Integer from 0 to 3 if the register type is **T**
- Integer from 0 to 15 if the register type is **L**
- Integer from 0 to 64 if the register type is **M**
- Integer from 0 to 4 if the register type is **Y**

Remark: The depth usage for the X registers is particular: from an application's point of view (ComET terminal or EDI application) X registers depths 0 and 1 point to the same value. Either depth can be set and the value is reflected on the other. Depths 2 and 3 are independent. When used in Compiled Sequences, when no depth is specified or if 0 is specified, it is the thread number in which the register is accessed that is used as the register depth.

Example:

```
*K717:1; // Read the value of the parameter K717 of the UltimET Light at the depth 1
```

4.2.3 Register value writing

The «*» sign in front of a register enables the identification of the UltimET Light as the addressee. This sign has the same meaning than '.axis' in the AccurET position controller syntax. The combination of both notations is possible to refer to several addressees (refer to the example below).

Here is the syntax allowing the user to change the value in a register.

Syntax:

***<register> [:<depth>][.<axis>] [<operator>] = <P1>**

Fields put in 'square brackets' (like: [<operator>]) are **optional**. They are not always used.

<register> Defines the register used. It must include the **type** of register as well as its corresponding **number**:

Possible types:

- K, X, L, T, Y

| | |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | <p>Possible numbers:</p> <ul style="list-style-type: none"> Integer (refer to §4.2.1.1) |
| [:<depth>] | <p>Some registers may contain different values simultaneously. Each value is stored at a different depth. If no depth is defined, depth 0 is automatically used by default.</p> <p>Possible values:</p> <ul style="list-style-type: none"> Integer from 0 to 7 if the register type is K Integer from 0 to 3 if the register type is X Integer from 0 to 3 if the register type is T Integer from 0 to 15 if the register type is L Integer from 0 to 4 if the register type is Y |
| Remark: | <p>The depth usage for the X registers is particular: from an application's point of view (ComET terminal or EDI application) X registers depths 0 and 1 point to the same value. Either depth can be set and the value is reflected on the other. Depths 2 and 3 are independent. When used in Compiled Sequences, when no depth is specified or if 0 is specified, it is the thread number in which the register is accessed that is used as the register depth.</p> |
| [.<axis>] | <p>Axis or group of axis number whose registers need to be modified.</p> <p>Possible values:</p> <ul style="list-style-type: none"> Integer from 0 to 62 according to the number of axis if the command refers to one single axis. Symbol ! if the command refers to all linked axes. Some selected axes numbers between commas. Y (indirect parameterization) |
| [<operator>] | <p>Mathematical sign for arithmetic and logical operations. It is available only for K and X registers.</p> <p>Possible values:</p> <ul style="list-style-type: none"> + addition. - subtraction. * multiplication. / division. % modulo (not available for Float 32 and 64 bits). ~ bitwise not (not available for Float 32 and 64 bits). & bitwise AND (not available for Float 32 and 64 bits). bitwise OR (not available for Float 32 and 64 bits). ^ bitwise XOR (not available for Float 32 and 64 bits). &~ bitwise NOT AND (not available for Float 32 and 64 bits). ~ bitwise NOT OR (not available for Float 32 and 64 bits). ^~ bitwise NOT XOR (not available for Float 32 and 64 bits). >> arithmetic shift to the right (not available for Float 32 and 64 bits). << arithmetic shift to the left (not available for Float 32 and 64 bits). <p>Refer to §7.3.6 for more information.</p> |
| = | obligatory link sign. |
| <P1> | <p>Value of the register.</p> <p>Possible values:</p> <ul style="list-style-type: none"> Immediate value in Integer, Long, Float or Double. Value contained in another register at any depth (indirect value). It can be taken from any axis. <P1> syntax: <register_name>[:<depth>].<axis> <p>Like for commands, when assigning a register with another on the same axis, the UltimET will send the command as is; otherwise the UltimET will first fetch the actual value and send it in the assignment command.</p> |

Example:

```

*K177 = 3;           // The value 3 is attributed to the depth 0 of the parameter
                    // K177 of the UltimET Light.
*K177 = X21.2;       // The value of the user variable X21 of the axis 2 is
                    // attributed to the depth 0 of the parameter K177 of the
                    // UltimET Light.
*K177.(1,4,5) = 3;   // The value 3 is attributed to the depth 0 of the parameter
// K177 of the axes 1, 4 and 5 and to the UltimET Light.
*K177.! = 3;         // The value 3 is attributed to the depth 0 of the parameter
                    // K177 of all axes connected to the UltimET and to the
                    // UltimET Light.

X0.0 = 1050;
X0.0 %= 100;         // Modulo 100 => X0 = 50

```

The indirect parameterization (Y) allows the user to give the value of the parameter ***K198** to a register (X, K and L) when it is equal to Y or y.

| Parameter | Comment |
|-----------|------------------------------------------------------------------|
| *K198 | The register number is given by *K198 when it is equal to Y or y |

Example:

When the user writes *XY=0 (or *Xy=0) and *K198=10, then it corresponds to *X10=0.

Remark: For all registers a **maximum value** and a **minimum value** are defined. If a higher value than the maximum value or a smaller value than the minimum value is given to a register, the value will automatically be restricted by those two limits at any depth. A **default value** is also defined for each register but only for depth 0. The default value for the other depths is the same for all registers and corresponds to 0.

Some K parameters use different depths. The depths of a K parameter have always the same unit but different values. When performing interpolation, the depths often represent the different axes.

Example:

Depth 0 of *K545 is the speed limit for the axis X of the interpolation group 0.
 Depth 1 of *K545 is the speed limit for the axis Y of the interpolation group 0.
 Depth 2 of *K545 is the speed limit for the axis Z of the interpolation group 0.
 Depth 3 of *K545 is the speed limit for the axis Θ of the interpolation group 0.
 Depth 4 of *K545 is the speed limit for the axis X of the interpolation group 1.
 Depth 5 of *K545 is the speed limit for the axis Y of the interpolation group 1.
 Depth 6 of *K545 is the speed limit for the axis Z of the interpolation group 1.
 Depth 7 of *K545 is the speed limit for the axis Θ of the interpolation group 1.

4.2.4 CLX command

The **CLX** command (**CL**ear **X** variables) clears (resets) all X (X, XF, XL and XD) and Y registers according to the <P1> value.

| Command | Bit# | <P1> value | Comment |
|----------------------|------|------------|---------------------------|
| *CLX[.<axis>] = <P1> | 0 | 1 | Clears X user variables. |
| | 1 | 2 | Clears XF user variables. |
| | 2 | 4 | Clears XL user variables. |
| | 3 | 8 | Clears XD user variables. |
| | 4 | 16 | Clears Y registers. |

Remark: The field put in 'square brackets', [.<axis>] is optional.
 If no bit is set (i.e. <P1> = 0), all X (i.e. X, XF, XL, XD) and Y user variables are cleared!

Example:

```
*X2 = 12;           // The *X2 user variable contains the value 12.
*XF4 = 20;          // The *XF4 user variable contains the value 20.
*CLX = 1;           // The *XF4 variable (as well as all the others XF variables)
                    // contains the value 0.
```

4.2.5 SET_RANGE command

The **SET_RANGE** command sets in one block n-1 values into X, XL, XF, XD or L registers. For X, XF and L registers, the maximum number of values is n = 127. For XL and XD registers, the maximum number of values is n = 100. It sets from a specified index at a fixed depth (<P1> value) the values in the specified type of register.

| Command | <Px> | Comment |
|------------------------------|------|---------------------------------------------------------------------------------------------|
| *SET_RANGE = <P1>,...,<Pn+1> | 1 | Type of register, first index and depth from where the different values have to be written. |
| | 2 | Value 1 to be written in the register specified in parameter 1. |
| | ... | ... |
| | n+1 | Value n to be written in the register specified in parameter 1. |

Example:

```
*SET_RANGE = *X1:2, 55, 56;    // Values 55 and 56 are set in X1:2 and X2:2
```

4.2.6 CHKDISTGRT command

The **CHKDISTGRT** command monitors the difference between two specified registers to ensure it is greater than a given value.

| Command | <P1> | <P2> | <P3> | Comment |
|----------------------------|------------------------|------------------------|-------------------------------------------|----------------------------------------------|
| *CHKDISTGRT=<P1>,<P2>,<P3> | Register specification | Register specification | Register specification or immediate value | Error 5800 is raised if <P2> - <P1> < <P3> |

The command compares the values of the first two parameters which must represent register specifications and generates the error 5800 if its absolute value is smaller than the value specified by the third parameter, which can be either an immediate value or another register specification. All parameters must have the same value type (long, float, etc.).

The comparison is performed every TransnET cycle, in the high priority interrupt. It starts immediately after the command is issued. Specifying an immediate null value as the third parameter disables the check. Disabling the feature is required before changing any parameter on the fly.

The feature is primarily implemented to monitor the positions of dual gantries, read from RTV monitorings, to ensure they remain safely apart.

Examples:

```
*CHKDISTGRT = X1:2,X2:2,1000
*CHKDISTGRT = XF2,XF3,XF4
*CHKDISTGRT = ML450,ML451,1000L
```

4.3 Communication with the other nodes

4.3.1 Command for another node

From an UltimET Light sequence, it is possible to send commands to the other nodes present on the TransnET. To do so, the command must be written in the sequence by specifying the wanted node as the addressee.

Example:

```
MVE.3 = 1000000L;    // Ask the motor of the node 3 to go to the position 1000000
```

4.3.2 Parameter coming from another node

The use of another node's variable as a parameter of a command is possible. The UltimET Light will first take the parameter's value in the specified node, and secondly, it will replace the parameter with the value before sending the command. This process is within the UltimET Light and totally invisible for the user.

Example:

```
MVE.2 = X1.4;           // Ask the motor of the node 2 to go to the position given
                        // by the variable X1 of the node 4.
```

When a command is intended to several nodes with one variable, each node can use its own variable. To do so, the node of the parameter must not be specified.

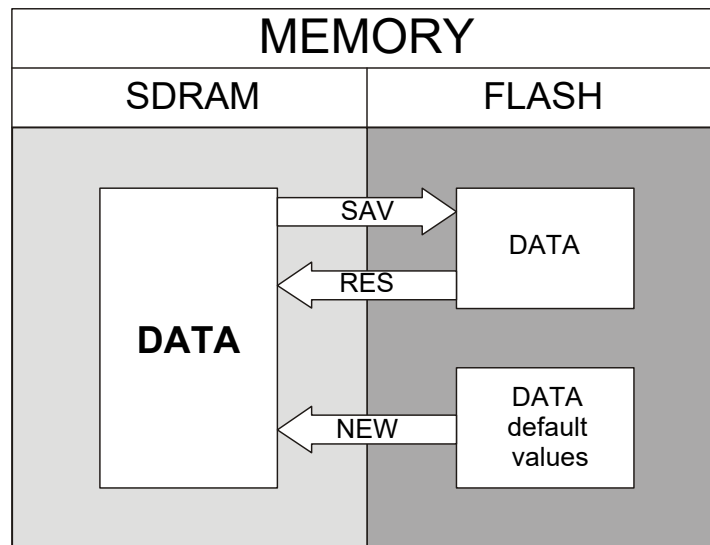
Example:

```
MVE.(1,2,3) = X10;      // The nodes 1, 2 and 3 execute a movement with the value of
                        // the variable X10 as destination.
```

4.4 Save the settings

The UltimET Light has **SDRAM memory** and **flash memory**. The SDRAM is a volatile memory which is erased each time the UltimET Light is switched off, whereas the flash is a non-volatile memory and the data stored inside are not lost when the UltimET Light is switched off. **All calculations and operations done by the controller are realized with the values present in the SDRAM.**

SAV, RES and NEW commands allow the user to transfer data from the SDRAM to the flash and vice-versa.



The **SAV** command (**SAVe**) **saves into the 'flash'** memory the UltimET Light's data (K, X, L and Y registers) as well as the sequence so that they are not lost when the controller is switched off and on again. The type of saved data is defined by the parameter's value of the command.

| Command | <P1> | Comment |
|----------------------|------|---------------------------------------------------------------------------------------------------------------------------|
| *SAV[.<axis>] = <P1> | 0 | Saves sequence, L registers (Look-up table), K registers (parameter), X and Y register's (user variable) in flash memory. |
| | 1 | Saves sequence and L registers (Look-up table) in flash memory. |
| | 2 | Saves K registers (parameter), X and Y registers (user variable) in flash memory. |
| | 3 | Saves K registers (parameter) in flash memory. |
| | 5 | Saves X and Y registers (user variable) in flash memory. |
| | 6 | Saves L registers (Look-up table) in flash memory. |
| | 7 | Saves sequence in flash memory. |

| Command | <P1> | Sequence | L | K KF KL KD | X XF XL XD | Y |
|------------------|------|----------|---|------------|------------|---|
| *SAV.<axis>=<P1> | 0 | x | x | x | x | x |
| | 1 | x | x | | | |
| | 2 | | | x | x | x |
| | 3 | | | x | | |
| | 5 | | | | x | x |
| | 6 | | x | | | |
| | 7 | x | | | | |

The **RES** command (**REStore**) restores into the **SDRAM** memory the UltimET Light's data (K, X, L and Y registers) as well as the sequence previously saved with SAV command into the "flash" memory. The type of restored data is defined by the parameter's value of the command.


| Command | <P1> | Comment |
|----------------------|------|----------------------------------------------------------------------------------------------------------------------------------------|
| *RES[.<axis>] = <P1> | 0 | Restores sequence, L registers (Look-up table), K registers (parameter), X and Y registers (user variable) from flash to sdram memory. |
| | 1 | Restores sequence and L registers (Look-up table) from flash to sdram memory. |
| | 2 | Restores K registers (parameter), X and Y registers (user variable) from flash to sdram memory. |
| | 3 | Restores K registers (parameter) from flash to sdram memory. |
| | 5 | Restores X and Y registers (user variable) from flash to sdram memory. |
| | 6 | Restores L register's (Look-up table) from flash to sdram memory. |
| | 7 | Restores sequence from flash to sdram memory. |

| Command | <P1> | Sequence | L | K KF KL KD | X XF XL XD | Y |
|----------------------|------|----------|---|------------|------------|---|
| *RES[.<axis>] = <P1> | 0 | x | x | x | x | x |
| | 1 | x | x | | | |
| | 2 | | | x | x | x |
| | 3 | | | x | | |
| | 5 | | | | x | x |
| | 6 | | x | | | |
| | 7 | x | | | | |

The **NEW** command **reloads** in the **SDRAM** the default values of K registers and **clears** the X, L and Y registers and the sequence stored in the SDRAM, depending on the parameter's value of the command.

| Command | <P1> | Comment |
|----------------------|------|------------------------------------------------------------------------------------------------------------------------|
| *NEW[.<axis>] = <P1> | 0 | Clears the X and Y registers (user variable), sequence, sets default values for K registers (parameter) in ram memory. |
| | 1 | Clears sequence in ram memory. |
| | 2 | Sets default values for K registers (parameter) in ram memory. |
| | 3 | Sets default values for K registers (parameter) in ram memory. |
| | 5 | Clears X and Y registers (user variables) in sdram memory. |
| | 6 | Clears L registers (Look-up table) in ram memory. |
| | 7 | Clears sequence in ram memory. |

| Command | <P1> | Sequence | L | K KF KL KD | X XF XL XD | Y |
|----------------------|------|----------|---|------------|------------|---|
| *NEW[.<axis>] = <P1> | 0 | x | | x | x | x |
| | 1 | x | | | | |
| | 2 | | | x | | |
| | 3 | | | x | | |
| | 5 | | | | x | x |
| | 6 | | x | | | |
| | 7 | x | | | | |

| | |
|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | After executing RES and NEW commands, SDRAM ordinary values are replaced by the values read in the 'flash' memory and are definitively lost. Similarly, the SAV command crushes the values contained in the 'flash' with those contained in the SDRAM. |
|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Remark: The field put in 'square brackets', [.<axis>] is **optional**.

When the UltimET Light is **switched on**, data is automatically restored, like with a ***RES = 0** command (so that it is not necessary to do it manually).

All parameters K depths are saved and read when SAV, RES and NEW commands are executed. Moreover, these commands are generally not used in a sequence but only when sending on-line commands or with ComET.

This is possible to specify the registers and sequence versions used in each controller. The configuration of the version number is done in the ComET editor before the download of a sequence or registers to the controller. The file header contains the version of each register type present in the file.

Header example:

```
# COMET special parameters
# ISO linear units: m, m/s, m/s2, V, A, s
# ISO rotary units: t, t/s, t/s2, V, A, s
# ETEL Parameters Upload
# Written by "edi-tra" v2.00A
# Date: Thu Apr 30 14:09:08 2009
# User K,KL,KF,KD register version axis * : 1.01A
# User X,XL,XF,XD register version axis * : 2.01B
# User LD look-up table version axis * : 1.00C
```

In order to set this version number, the user must either modify the current header (if present) or create a new one by using ComET editor function called 'Insert version header' (edit menu) and customize it.

The monitoring ***M282** allows the user to know the registers or sequence version.

| Monitoring | Comment |
|--------------|------------------------------------------------------------------------------------------|
| *M282 | Registers or sequence version (depth 0 to 2 : not used, depth 3 to 7 as for SAV command) |

4.5 UltimET Light reset

It is sometimes required to reboot the UltimET Light, typically after a firmware download to take it into account. The **RSU** command is provided to that effect.

| Command | <P1> | Comment |
|-----------------|--------------------------------------------------|------------------------------------|
| *RSU=255 | Value = 255 Mandatory authorization parameter | Reboot the UltimET Light processor |

The ***RSU** command reboots the board's processor, meaning that:

- the application will lose the connection to the board.
- the TransnET will drop, making all the axes generate an error.

The host application must reopen the connection on the UltimET Light after an ***RSU** command, and if needed wait for all the expected axes to be present once again and clear the "TransnET down" errors.

Remark: On the UltimET Light PCIe, if the FPGA has changed in the download, the ERROR_BAD_FPGA_VERSION error (*M64=4000) will be raised to notify that in this case, the ***RSU** command will not be enough to take into account the new FPGA and then a complete shutdown/restart of the host PC is required.

4.6 UltimET Light software characteristics

The following monitorings indicate useful data for the user (they cannot be modified):

| M | Alias | Value | Comment |
|-------------|-------|-------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *M70 | - | 48 | Indicates the type of UltimET Light |
| *M71 | - | - | Gives the software boot version of UltimET Light. A special ETEL procedure allows the conversion of this value in the software boot version (format is similar to M72) |

| M | Alias | Value | Comment |
|------|-------|-------|---------------------------------------------------------------------------------------------------------------------------------------------|
| *M72 | VER | - | Gives the firmware version of the UltimET Light. A special ETEL procedure allows the conversion of this value in the firmware version |
| *M73 | SER | - | Gives the serial number of the UltimET Light |
| *M85 | - | - | Gives the article number string. The 18 strings of the article number are read using 5 depths of M85. Each depth shows 4 strings (in ASCII) |

The monitoring M72 can be directly read with the command VER. The UltimET Light sends a firmware version, as a hexadecimal number, under the following form:

*VER = 0x**WWWXXYY**, with the following possible values:

| | WWW: version # (3 digits) | | | | XX: revision index (2 digits) | | | | | | | | | | YY: status (2 digits) | |
|-----------------|---------------------------|-----|------|-----|-------------------------------|------------|-----|-----------|-----------|-----|----|----|----|-----|-----------------------|---------|
| Values examples | 100 | ... | 120 | ... | 00 | 01 | ... | 40 | 41 | ... | 80 | 81 | 82 | ... | 00 | 80 |
| Meaning | 1.00 | ... | 1.20 | ... | α_0 | α_1 | ... | β_0 | β_1 | ... | A | B | C | ... | released | in dev. |

For example, the hexadecimal value 0x**120800** means that the firmware version 1.20A, released, is in the UltimET Light.

4.7 Synchronized homing on several axes

The **IND** command allows the user to manage a synchronized homing procedure on several axes (useful in particular for gantry system in level 1 ONLY).

| Command | <P1> | <P2> | Comment |
|------------------|------------------------------------------|-----------------------------|---------------------------------------------|
| *IND=<P1>[,<P2>] | Mask of the axes concerned by the homing | Disable adjustment movement | Starts a homing sequence on a group of axes |

All the axes specified in the mask start to move at the same time and stop when each axis has found its own reference mark. At the end of the synchronized homing process, the UltimET Light automatically sends a MVE command (MVE.mask_of_home_sync_axes=ML6.first_of_home_sync_axes). So all the axes take into account the zero position adjustment given by their own KL45 and are then correctly aligned. This adjustment movement can be disabled by using <P2>=255. Any other value or no value of <P2> enables it. If one axis meets a limit switch or a mechanical end stop before the reference mark, all axes break together and go in the opposite direction. This procedure avoids any mechanical deformation.

The homing parameters of the position controllers (K40, KL41, KL42, KL46, KL47 and KL48) must be identical for the axes concerned by the homing (refer to the corresponding '**Operation & Software Manual**' for more information about the homing parameters of the controllers).

The position controller's homing modes supported by the synchronized homing of the UltimET Light are:

| K40 | Direction | Mechanical end stop | Home switch (DIN2) | Limit switch (DIN) K58=1 | Limit switch (L1/L2) K58=2 | Limit switch (L/H) K58=3 | Mono-ref. mark encoder | Multi-ref. mark encoder | Remark |
|-----|-----------|---------------------|--------------------|--------------------------|----------------------------|--------------------------|------------------------|-------------------------|------------------------------------------------------------------------|
| 8 | + | X | | | | | OK | | |
| 9 | - | X | | | | | OK | | |
| 10 | + | | | X | X | X | OK | | |
| 11 | - | | | X | X | X | OK | | |
| 12 | + | X | | | | | | OK | |
| 13 | - | X | | | | | | OK | |
| 38 | + | X | | | | | OK | | The trip is defined by KL46 after having found the mechanical end stop |

| K40 | Direction | Mechanical end stop | Home switch (DIN2) | Limit switch (DIN) K58=1 | Limit switch (L1/L2) K58=2 | Limit switch (L/H) K58=3 | Mono-ref. mark encoder | Multi-ref. mark encoder | Remark |
|-----|-----------|---------------------|--------------------|--------------------------|----------------------------|--------------------------|------------------------|-------------------------|--------------------------------------------------------------------------------|
| 39 | - | X | | | | | OK | | The trip is defined by KL46 after having found the mechanical end stop |
| 44 | + | | | X | X | X | OK | | |
| 45 | - | | | X | X | X | OK | | |
| 46 | | | | | | | | | Homing without effect on the position. It is used with absolute EnDat encoder. |

Different error messages may occur during a synchronized homing. Refer to [§12.](#) for more information these errors.

Example with a gantry system:

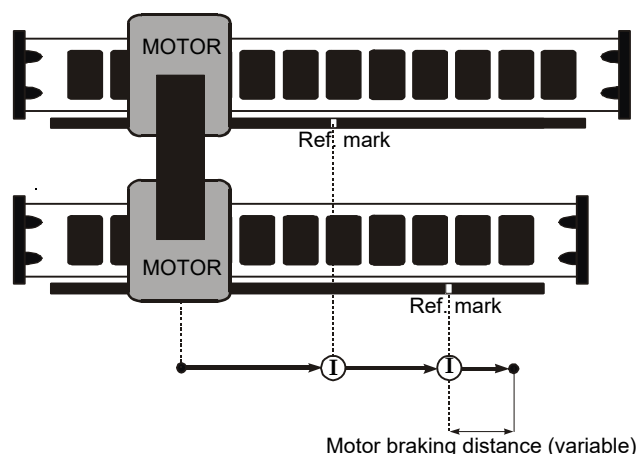
Here is a description of the symbols used in this paragraph:

- ① Reference position
- Reference mark
- ② Zero machine
- Motor position
- Motor trip

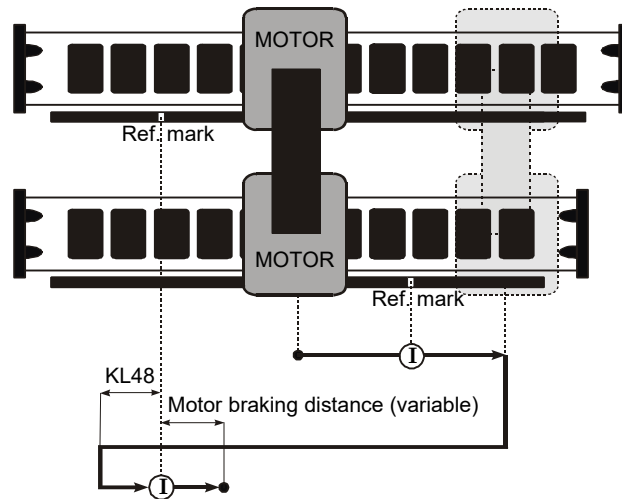
K40 = 8:

Homing with a mono-reference mark with a positive movement. To have the mono-reference mark always in the same position, the motor must find it when moving in the determined direction. There are three possibilities:

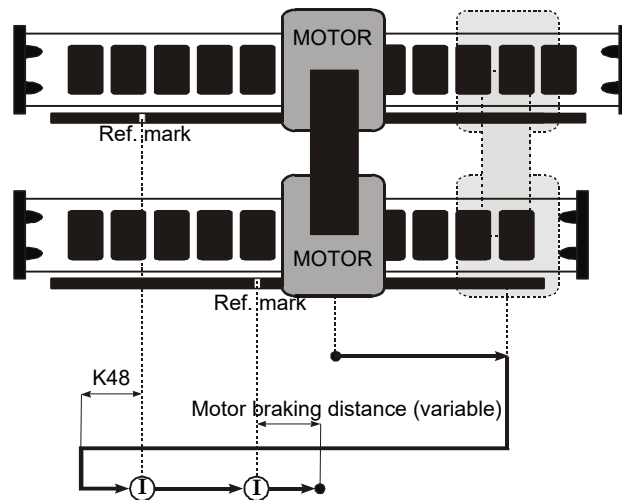
- Both motors are on the left of the mono-reference marks at the beginning of the homing. They move and meet successively both reference marks



- Only one motor is on the left of the reference mark at the beginning of the homing. They move and one motor meets its reference mark. They both continue to move. After having found the mechanical end stop, they both come back. When the second meets its reference mark (in the wrong direction), they keep moving the distance given by the parameter KL48 before changing direction a second time to find the reference mark in the right direction.



3. Both motors are on the left of the mono-reference marks at the beginning of the homing. They move and after having found one mechanical end stop, they both come back. When they both meet their corresponding mono-reference mark (in the wrong direction), they keep moving the distance (after the second mono-reference mark) given by the parameter KL48 before changing direction a second time to find their reference mark in the right direction.



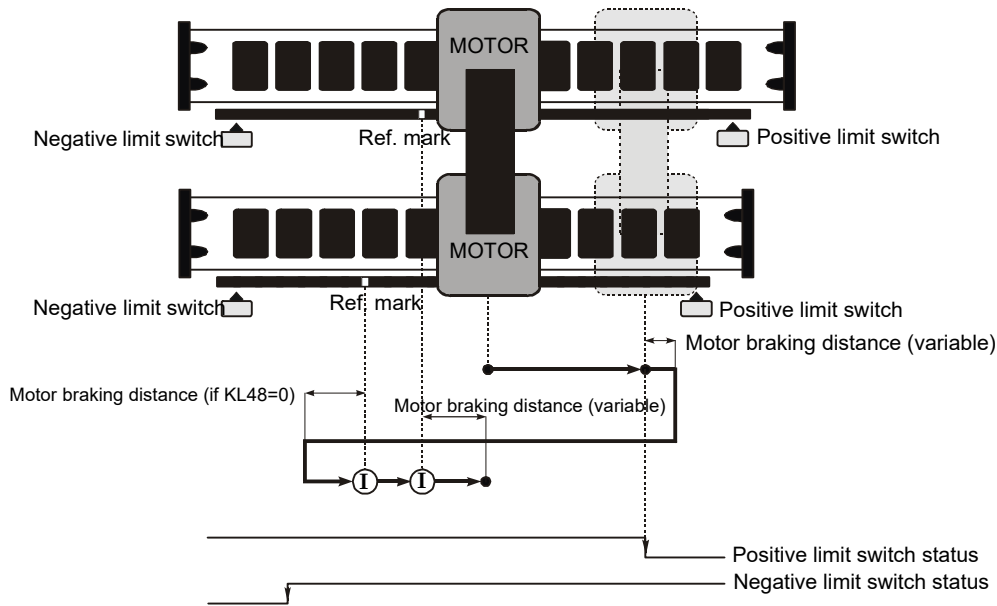
K40 = 9:

Same as K40 = 8 but with a negative movement.

K40 = 10:

Same as K40 = 8 but the motor changes direction when meeting a limit switch instead of mechanical end stop. Only case 3 (of K40=8) is shown.

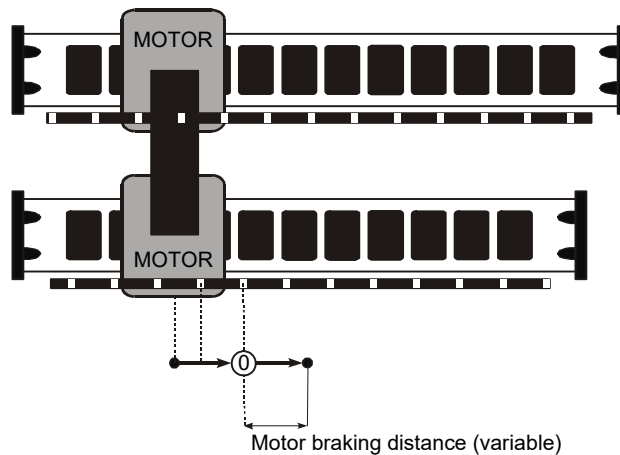
If the motor is on the limit switch, it moves back the distance given by the parameter KL48.

**K40 = 11:**

Same as K40 = 10 with a negative movement.

K40 = 12:

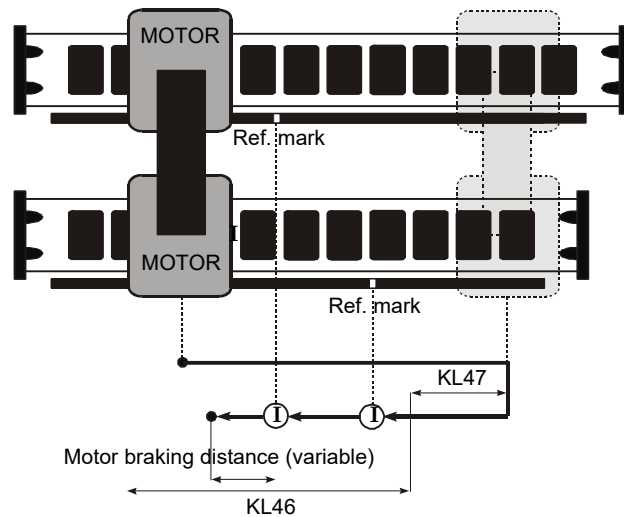
Homing with a multi-reference mark with a positive movement. The motors need to find only two successive reference marks to determine its **absolute position** (the '0 machine' is always positioned at the same place regardless the two reference marks found). If one motor finds a reference mark followed with a mechanical end stop (K40 = 12 or 13), the reference mark is not considered and the motor starts the homing in the opposite direction.

**K40 = 13:**

Same as K40 = 12 but with a negative movement.

K40 = 38:

Homing on a mono-reference mark but only after having found the mechanical end stop. Once found, the motor moves back the distance given by the parameter KL47 and finds the reference mark within the distance given by the parameter KL46.

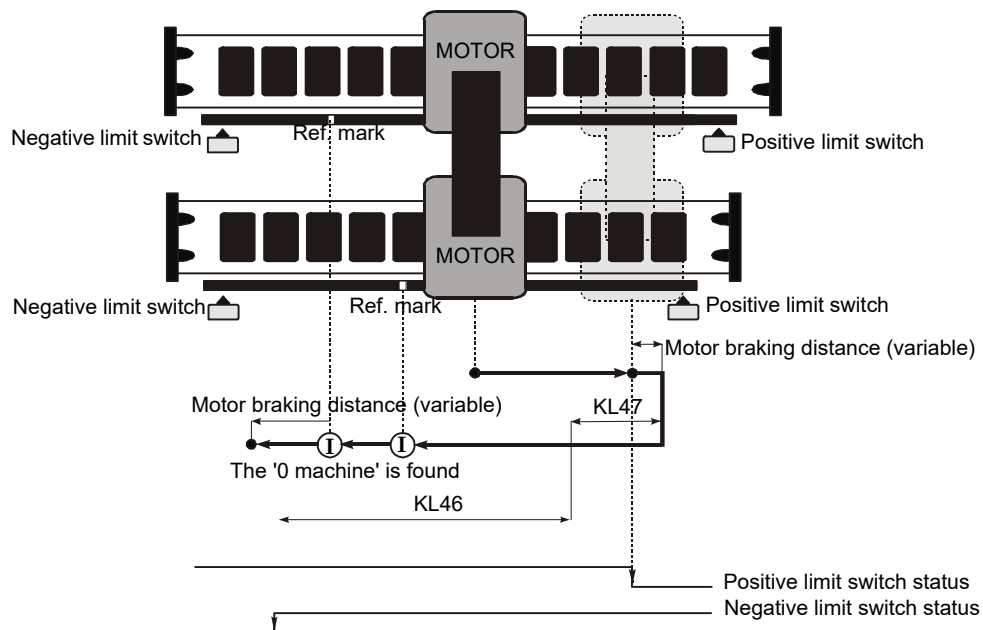


K40 = 39:

Same as K40 = 38 with a negative movement.

K40 = 44:

Homing on a mono-reference mark but only after having found the limit switch. Once found, the motor moves back the distance given by the parameter KL47 and finds the reference mark with the distance given by the parameter KL46. If the index is not found within the distance given by the parameter KL46, the **SING IDX SEARCH** error (M64=62) appears (the distance given by the parameter KL46 must be smaller than the one between both limit switches).



K40 = 45:

Same as K40 = 44 with a negative movement.

K40 = 46:

Homing without effect on the position. It is used with absolute EnDat encoder in order to not change the position. The offset for absolute position (KL45) has no effect in that case.

4.8 WAIT command

The commands 'wait' enable the waiting for an event. The command is active as long as the condition defined by this command is not fulfilled: it is a blocking command.

Each thread of the UltimET Light is independent. If a waiting is sent on a thread, only this one becomes busy and the others continue to work normally. The monitoring ***M101** allows the user to know the mask of the thread currently in waiting state.

| Monitoring | Comment |
|--------------|------------------------------------------------|
| *M101 | Mask of the thread currently in waiting state. |

Example of a command 'wait' to wait for the end of a movement in a sequence:

```
MVE.0 = 1000L;           // Starts the movement
WTM.0;                   // Waits for the end of the movement of the node 0
MVE.1 = 2000L;           // Starts a movement on the node 1.
```

4.8.1 Wait command list

In the following list, <P1> and <P2> represent the parameters of the command 'wait'. It can be any registers, the accumulator or an immediate value.

Warning: If <P1> and <P2> are registers with a value which can change, they must belong to the node executing the command 'wait'. This command will read only once the parameters on a node at the beginning of the waiting. Then, it will not be able to see the changes of the register's value.

For example, *WPG = X1.1,12 is forbidden, the command 'wait' is done on the UltimET Light but the variable X1.1 is on the node1.

4.8.1.1 WTT command

The **WTT** command (**WaIT Time**) makes a pause in the sequence progression. The pause duration is set with the parameter of the command.

| Command | <P1> | Comment |
|-----------------------------------------|--------------|---------------------------------------------------------|
| *WTT[.<axis>] = <P1> | Waiting time | Wait the time defined by <P1>, in increments of 100 µs. |

Remark: Refer to [§4.8.1.6](#) to know how to clear a 'wait' command.

Example:

The interrupt time is equal to 100 ms.
 *WTT = 5000; // The pause duration is equal to 5000 × 100 ms = 500 ms.

4.8.1.2 WTM commands

The **WTM** command (**WaIT for end of Movement**) allows the user to wait for the end of the interpolated movement before going on with the execution of the sequence.

| Command | <P1> | Comment |
|-----------------------------------------|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *WTM[.<axis>] = <P1> | Interpolation group number | Waits for the end of the interpolated movement before continuing the execution of the sequence. This condition is fulfilled when the current movement is finished and the buffer containing the command of interpolation is empty. |

Remark: Refer to [§4.8.1.6](#) to know how to clear a 'wait' command.

Example:

```

...
*ILINE = 0,1000,2000,0,0; // Definition of the interpolated movement on the group 0
*ILINE = 0,2000,1000,0,0;
*WTM = 0; // Wait for the end of the movement described above

```

Remark: If the motor is not moving when the WTM command is sent, it is ignored.

4.8.1.3 Wait on bits: WBS, WBC, WSBS and WSBC commands

WBS (Wait Bit Set) and **WBC (Wait Bit Clear)** commands test one or several bits of X, K or M registers and go on with the execution of the sequence if the bits are set to 1 (WBS) or to 0 (WBC). The bits are numbered from the right to the left from 0 to 63.

| Command | <P1> | <P2> | Comment |
|---------------------------|------------------|------------|---------------------------------------------------------------------------------------------------------------------------------|
| *WBS[.<axis>] = <P1>,<P2> | Register to test | Mask value | Waits for the specified bits (which have their mask value included in <P2>) of the specified register <P1> to be all set. |
| *WBC[.<axis>] = <P1>,<P2> | Register to test | Mask value | Waits for the specified bits (which have their mask value included in <P2>) of the specified register (<P1>) to be all cleared. |

The field <P2> must contain the **mask** that selects the bits to be tested in the register included in the field <P1>. This mask is obtained by transforming in binary the value contained in <P2>. The bits with 1 are those tested by the WBS and WBC commands.

Example:

```

*WBS.0 = DIN.0,0x04; // Wait for the digital output 3 (numbered from 1) of the
// node 0 to be at 1.
*WBC = *X3:2,0x0F; // Wait for the bits 0, 1, 2 and 3 of the UltimET Light's
// variable *X3 of the thread 2 to be at 0.

```

WSBS (Wait Status Bit Set) and **WSBC (Wait Status Bit Clear)** commands test one or several bits of M63 monitoring and go on with the execution of the sequence if the bits are set to 1 (WSBS) or to 0 (WSBC). The bits are numbered from the right to the left from 0 to 63.

| Command | <P1> | <P2> | Comment |
|-------------------|------------------|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *WSBS = <P1>,<P2> | Bits mask of M63 | Mask of the node | Wait for the status bits of M63 (which have their mask value included in <P1>) to be set in several axes (which have their mask value included in <P2>) at the same time. |
| *WSBC = <P1>,<P2> | Bits mask of M63 | Mask of the node | Wait for the status bits of M63 (which have their mask value included in <P1>) to be cleared in several axes (which have their mask value included in <P2>) at the same time. |

Example:

Switch on the digital output 1 of node 2 when the nodes 3 and 6 are in window

```

MVE.3 = 53840L;
MVE.6 = 121400L;
*WSBS = 0x00200000,0x48L;
DOUT.2 = 1; // Activation of DOUT1 of the node 2

```

Switch on the digital output 1 of node 4 when the sequences of the node 2 and 3 are finished.

```

JMP.(2,3) = 10; // Start sequence on nodes 2 and 3
*WSBC = 0x01000000,0xCL;
DOUT.4 = 1; // Activation of DOUT1 of the node 4

```

Remark: Refer to [§4.8.1.6](#) to know how to clear a 'wait' command.

4.8.1.4 Wait on values: WPL and WPG commands

WPL (Wait Parameter Lower than) and **WPG** (Wait Parameter Greater than) command allow the user to wait for the register specified in <P1> to be lower respectively greater than the value given in <P2> to continue the sequence execution.

| Command | <P1> | <P2> | Comment |
|---------------------------|------------------|-------------------|-------------------------------------------------------------------|
| *WPL[.<axis>] = <P1>,<P2> | Register to test | Value of register | Waits for the register <P1> to be lower than the value of <P2>. |
| *WPG[.<axis>] = <P1>,<P2> | Register to test | Value of register | Waits for the register <P1> to be greater than the value of <P2>. |

Remark: This function is not only dedicated to the movements but also to all K, M or X registers. If the values specified in <P2> are never reached by the chosen register, the pause has an infinite duration. If the condition is already met when the command is executed, the sequence goes on immediately.

The value in <P2> is only taken into account when the WPL and WPG commands are executed. Refer to [§4.8.1.6](#) to know how to clear a 'wait' command.

4.8.1.5 Wait on marks: WTK command

The **WTK** (WaiT for marK) command allows the user to wait for the execution of a mark from the trajectory calculator.

| Command | <P1> | <P2> | Comment |
|---------------------------|---------------------|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *WTK[.<axis>] = <P1>,<P2> | Interpolation group | Number of the mark | This command waits for the execution of a mark from the trajectory calculator. It enables the synchronization of a sequence with a movement. When the trajectory calculator will meet the specified mark, the waiting will be finished. |

Remark: Refer to [§4.8.1.6](#) to know how to clear a 'wait' command.

Example:

```

...
*ILINE = 0,500,2000,0,0; // Definition of an interpolated movement
*IMARK = 0,14,0,0; // Definition of a mark number 14
*ILINE = 0,1000,2000,0,0; // Definition of an interpolated movement
*ILINE = 0,2000,2000,0,0; // Definition of an interpolated movement
*WTK = 0,14; // Waiting for the mark number 14
DOUT.0 = 1; // Activation of the DOUT 1 of the node 0
...

```

Refer also to [§6.6.2](#) for more information about the WTK command.

4.8.1.6 Clear wait commands

All the 'wait' commands described in [§4.8.1](#) temporarily stop the execution of other normal commands on the same entry (of the 'wait' command) during all the waiting time. During this waiting time, the entry is in a busy state. The **CLRWAIT** command allows the user to clear a pending 'wait' which is blocking an entry specified by <P1>. On UltimET Light, the only possible entry we are concerned with is the Application Bus (PCI, PCIe or TCP/IP depending on the product). It is explicitly forbidden trying to cancel a 'wait' that is blocking a sequence thread (Bit#1 to 3 of *M101) otherwise an error will occur.

| Command | Comment |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *CLRWAIT.<axis>=<P1> | Clear busy state due to a wait command. The definition of <P1> is the same as for the monitoring M101 (Wait entry; refer to §4.8.1). The bit#5 to 7 which are reserved must be set to 0. |

The CLRWAIT.<axis> command could be sent as a normal command as well as an urgent command. The advantage of the urgent command is that this command will not be blocked by a previous pending 'wait' command (if present) on the same entry. The command *CLRWAIT has no effect if it is inside a compiled sequence.

4.9 User LEDs

The LEDs number 3 and 4 of the UltimET Light PCI / PCIe (refer to [§3.1.4](#) for the location) can be programmed by the user with the parameter ***K595**.

| Parameter | Comment |
|--------------|---------------------------------------------------------------------------------|
| *K595 | Control of the user's LED (1=on, 0=off, depth 0 = green led, depth 1 = red led) |

4.10 Local digital inputs / outputs

Remark: There is no local analog I/Os

4.10.1 Digital inputs

The monitorings ***M50** allows the user to read the status of the standard digital inputs (2 on the TCP/IP version and 4 on the PCI/PCIe version). The DIN command is an alias of this monitoring and uses the same syntax.

| M | Alias | Name | Comment |
|-------------|-------|------------|------------------------------------------------------------------------------|
| *M50 | DIN | DIN status | Gives the status of the digital inputs: DIN1 (bit# 0),..., to DIN4 (bit# 3). |

A bit equal to 1 means that the corresponding digital input is activated and equal to 0 means that the corresponding digital input is deactivated. Refer to [§3.1.2](#) for more hardware information.

Example:

Here is explained the use of the digital inputs bits values:

| DIN # | 4 | 3 | 2 | 1 |
|-------------------------------|---|---|---|---|
| Bit # | 3 | 2 | 1 | 0 |
| Decimal and hexadecimal value | 8 | 4 | 2 | 1 |

A simple conversion in binary of the value shown by DIN (in hexadecimal) is enough to know which digital inputs are activated and deactivated.

| DIN values | DIN4 | DIN3 | DIN2 | DIN1 |
|------------|--------|--------|--------|--------|
| | Bit# 3 | Bit# 2 | Bit# 1 | Bit# 0 |
| 3 | 0 | 0 | 1 | 1 |
| 11 | 1 | 0 | 1 | 1 |
| Hexa. | Binary | | | |

In the first line of the above-mentioned example, DIN1 and DIN2 are activated. The request ***DIN** gives:

```
*DIN = 3; // Reads the digital inputs state. The UltimET Light gives the value
           // 3 in hexa, because 0x3 in binary is 0011 and each bit represents
           // one of the digital inputs, from right to left.
```

In the second line of the above-mentioned example DIN1, DIN2 and DIN4 are activated. The request ***DIN** gives:

```
*DIN = 11; // Reads the digital inputs state. The UltimET Light gives the value
            // 11 in hexa, because 0x11 in binary is 1011 and each bit represents
            // one of the digital inputs, from right to left.
```

4.10.1.1 Protection signals on DIN

The bit #4 of the parameter *K33 allows the user to enable an external error if the status of the digital inputs matches with the mask defined in the parameter *K305.

| K | Name | Value | Bit # | Comment |
|------|------------|-------|-------|--------------------------------------------------------------|
| *K33 | Input mode | 16 | 4 | Enables external error if the DIN match with the mask *K305. |

The parameter *K305 defines the mask of the digital inputs to enable the external error.

| K | Name | Comment |
|-------|------------|---------------------------------------------------------------|
| *K305 | Input mask | DIN mask for external error. Bit 0-15: set, bit 16-31: reset. |

4.10.2 Digital outputs

The parameter *K171 allows the user to activate or deactivate the standard digital outputs (2 on the TCP/IP version and 4 on the PCI/PCIe version). The DOUT command is an alias of this parameter and uses the same syntax.

| K | Alias | Name | Comment |
|-------|-------|------------|-------------------------------------------------------------------------------------|
| *K171 | DOUT | DOUT value | Activates / deactivates the digital outputs: DOUT1 (bit# 0),..., to DOUT4 (bit# 3). |

A bit equal to 1 means that the corresponding digital output is activated and equal to 0 means that the corresponding digital output is deactivated. Refer to [§3.1.2](#) for more hardware information.

The status of the DOUT can be read with the monitoring *M171:

| M | Name | Comment |
|-------|-------------|-------------------------------------------------------------------|
| *M171 | DOUT status | Gives the state of the digital outputs (reflects parameter *K171) |

Example:

Here is explained the use of the digital outputs bits values:

| DOUT # | 4 | 3 | 2 | 1 |
|-------------------------------|---|---|---|---|
| Bit # | 3 | 2 | 1 | 0 |
| Decimal and hexadecimal value | 8 | 4 | 2 | 1 |

A simple conversion in binary of the value shown by the monitoring *M171 (in hexadecimal) is enough to know which digital inputs are activated and deactivated.

| DOUT values | DOUT4 | DOUT3 | DOUT2 | DOUT1 |
|-------------|--------|--------|--------|--------|
| | Bit# 3 | Bit# 2 | Bit# 1 | Bit# 0 |
| 5 | 0 | 1 | 0 | 1 |
| 7 | 0 | 1 | 1 | 1 |
| Hexa. | Binary | | | |

In the first line of the above-mentioned example, DOUT1 and 3 are activated.

```
*DOUT = 3; // Activates DOUT1 and 2, and deactivates DOUT3 and 4.
```

In the second line of the above-mentioned example, DOUT1 to 3 are activated.

```
*DOUT = 2; // Activates DOUT2, and deactivates DOUT1, 3 and 4.
```

The digital outputs can be dedicated to TransnET cycle time output. To do so, the parameter *K353 must be used. Each bit of this parameter can take the signal out or not on the corresponding digital output (bit0 = DOUT1, bit1 = DOUT2,...). It is possible to take the signal out from several different outputs at the same time.

| K | Name | Comment |
|-------|-------------------|------------------------------|
| *K353 | Mask for TransnET | DOUT mask for TransnET cycle |

The TransnET cycle time can be output every K360 cycle.

| K | Name | Comment |
|-------|-------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| *K360 | Mask for TransnET | Set on the DOUT one TransnET synchro signal every K360 cycles (0=no signal, 1=each TransnET cycle, 2= every 2 TransnET cycle,...) |

Remark: There is an error test to check if the same DOUTs are used by different parameters. This test takes into account the direct use by the parameter *K171, the use of the marks by the parameter *K541 as well as the use for the TransnET cycle by the parameter *K353. The parameter *K357 (refer to [§4.10.2.1](#)) is not managed by this test as this parameter covers the most important function concerning the inputs / outputs and erase all the others in case of error.

4.10.2.1 Protection signals on DOUT

When a motor is integrated in a complex machine and an error is detected in the UltimET Light, it is important to send a message about it to the rest of the machine so that the other elements can adequately react. This is possible with the digital outputs. The parameter *K357 allows the user to set or reset one or two digital outputs when an error occurs. The output(s) are chosen via the binary value of the parameter *K357.

| K | Name | Comment |
|-------|---------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| *K357 | Mask of the DOUT if error | Mask of the digital output (DOUT) that must be set or reset when the controller is in error. Bit 0-15: set, bit 16-31: reset. |

It also allows the user to know if this output is reserved for this functionality. If the same output is set and reset in the parameter *K357, an error will occur.

As soon as an error occurs on the UltimET Light, the digital outputs present in the parameter *K357 move to the programmed status and no modification can be done on these outputs as long as the error is present. When the UltimET Light is reset, the digital outputs take their initial status back or their new status if modifications occurred during the error. The digital outputs not modified by the parameter *K357 remain available and work normally.

4.11 External inputs / outputs

It is possible to use an external I/O (Inputs/Outputs) module from WAGO and its access from the UltimET Light is done through TCP/IP Ethernet link, Modbus protocol on UDP. Those external I/Os are accessible in UltimET as if they were local I/Os. UltimET automatically creates a local process image of I/Os and cyclically updates it. The UltimET allows the user to have access up to 128 DIN (digital inputs), 128 DOUT (digital outputs), 16 AIN (analog inputs) and 16 AOUT (analog outputs).

Remark: A maximum of 64 modules can be connected to the UltimET Light.
The ETEL software is not compatible with modules which include on the same module input and output.
ETEL is compatible with all the digital input modules with less or equal to 16 channels, with all the digital outputs modules with less or equal to 16 channels, with all the analog inputs modules with less or equal to 16 bits and with all the analog outputs modules with less or equal to 16 channels.
The connection of special modules like stepper controller, RS232 interface, encoder interface, counter and so on is not supported.

This function is available on the UltimET Light having the number 1 in the following product number EU-Lxx-x-x-1xxx-xx. In this version, the UltimET has a Media Access Control address (MAC address) which is assigned

to the Auxiliary Communication interface of UltimET ('Aux Com' connector). Refer to [§3.1.2.3](#) or [§3.3.2.4](#) for more information. The MAC address of the UltimET Light Ethernet connection is accessible with the monitoring ***M203** and the one of the external I/Os ethernet interface with the monitoring ***M707**.

| M | Alias | Name | Comment |
|--------------|----------|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| *M203 | - | Ethernet MAC address | MAC address of the UltimET Light TCP/IP Ethernet interface (depth 0: four higher digits, depth 1: eight lower digits) |
| *M707 | IOMACADR | Aux com MAC address | MAC address of external I/Os Ethernet interface, whose connector is referred as Aux com (depth 0: four higher digits, depth 1: eight lower digits) |

When using this feature, the UltimET cyclically updates I/O data to and from a WAGO module through an Ethernet connection. To respect the cyclic update time, special care has to be taken to network topology. The highly recommended solution is a point to point connection. Nevertheless, if a network configuration is requested, we advise to use a sub-network with a limited number of connections and to avoid broadcast communication. The use of this configuration with both UltimET and WAGO module on a company network will lead to hundreds of milliseconds update time (broadcast communication) and must not be used.

4.11.1 Configuration

To activate such a feature, both WAGO modules and UltimET need to be configured.

4.11.1.1 WAGO configuration

The WAGO modules must be configured through WAGO tools. Please refer to the WAGO documentation (<http://www.wago.com>) for more information. The protocol used is the protocol Modbus UDP and not Modbus on TCP. the following information must be configured: IP address (for example 192.168.1.0) and the Sub-net mask (for example 255.255.255.0). The configuration is made with the Wago tool: "WAGO Ethernet Setting" (available free of charge from www.wago.com). The MODBUS communication time-out (10s by default) must be set at least with the UltimET I/Os refresh rate (refer to [§4.11.1.2](#)).

4.11.1.2 UltimET configuration

The feature managing external I/Os is by default disabled. The following setting has to be done to activate it:

- Set the WAGO module IP address (***K722** or ***IOIPADR** alias) as defined in the module.
- Set the UltimET client IP address (***K721** or ***IPCLADR** alias). This address must be in the same network range as the WAGO module IP address (e.g. UltimET client IP address is 192.168.10.140, WAGO module IP address is 192.168.1.10, sub-net mask is 255.255.0.0).
- Set the module response timeout (***K724** or ***IOTIMEOUT** alias).
- Set the external I/Os sub-net mask (***K725** or ***IOSUBNET** alias).
- Set, if necessary for the application, the IP address of the default gateway (***K726** or ***IOGATEWY** alias). It is not useful for point-to-point connection.
- Set external I/Os feature parameter to 1 (***K727** or ***EIOEN** alias). This parameter is taken into account at the UltimET start-up. This activation state may be checked through the monitoring ***M698** (or ***IOENBLD** alias).
- Set the external I/Os refresh time (***K729** or ***IOREFRESH** alias). This value is in milliseconds. This parameter is set to 10 ms by default.
- Save UltimET parameters (***SAV=2**) and reboot the UltimET (by opening the UltimET PCI/PCIe version with reset option or by switching off / on the power of the UltimET TCP/IP version).

At this step, external I/Os are mapped in the UltimET but the outputs are not updated until the ***EIOSTA=1** command is issued.

| K | Alias | Name | Comment |
|--------------|-------------------|---------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| *K721 | *IPCLADR | UltimET IP address | External I/Os IP address of the UltimET TCP/IP client |
| *K722 | *IOIPADR | External I/O module IP address | External I/Os module server IP address (IP address of the WAGO module) |
| *K724 | *IOTIMEOUT | External I/Os module response timeout | External I/Os module response timeout (ms). If this parameter is not set or set to 0, the default value (1s) is taken. |

| K | Alias | Name | Comment |
|-------|------------|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *K725 | *IOSUBNET | External I/Os subnet mask | External I/Os subnet mask used by the TCP/IP external I/O module |
| *K726 | *IOGATEWY | External gateway IP address | External I/Os default gateway IP address |
| *K727 | *EIOEN | External I/Os feature enable | External I/Os feature enable (switch on all the features related to external I/Os). The new value is taken into account after the next switch on of the controller |
| *K729 | *IOREFRESH | External I/Os refresh rate | External I/Os refresh rate (ms) |
| *K734 | *IONBREP | External I/Os nb repetition | External I/Os number of repetition of UDP message before error |

To have a fully operational configuration with analog interfaces, some analog I/Os related parameters have to be set. As explained below with more details, analog I/Os are accessible in both raw (integer 32-bit) and converted (float 32-bit) formats. Conversion between raw data and converted data require the setting of some parameters. This setting is described in [§4.11.2.6](#) for the analog inputs and [§4.11.2.7](#) for the analog outputs, as well as the setting of each analog I/O resolution. The number of available I/Os is accessible through monitoring registers and is described in [§4.11.2](#).

4.11.2 Use and setting of external I/Os on WAGO

External I/Os are accessible as if they were UltimET local I/Os. It means they are stored in K parameters and M monitorings and they can be managed as any other UltimET registers. Analog inputs and outputs are accessible either as raw data (integer K and M registers) or as converted data (float KF and MF registers). Aliases of registers accessing external I/Os start with the 'E' letter, abbreviation for 'External' (e.g. accessing first 16-bit of external digital inputs is done through *EDIN0 register). Additionally to these basic accesses through K and M registers, I/O dedicated functions allowing the management (read, write) of a single I/O data, are provided for more user-friendliness in compiled sequences.

4.11.2.1 I/Os configuration check

The external I/Os features must be activated and the UltimET needs to be configured (IP link, IO update cycle time...). When the configuration parameters are set and the UltimET is rebooted (this feature needs a reboot so that *K727 parameter is taken into account), the feature activity status is accessible through *M698 monitoring (or *IOENBLD alias). This monitoring is set to 1 if the two following conditions are fulfilled:

- There is a valid MAC address for this external I/Os Ethernet interface (this information is stored in *M707)
- The feature is enabled, which means that *K727=1 and the system has been rebooted

Otherwise, the *M698 monitoring (or *IOENBLD alias) is set to 0.

| M | Alias | Name | Comment |
|-------|-----------|------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *M698 | *IOENBLD | External IO feature enabled status | Gives the external I/Os feature enabled status. |
| *M699 | *NMACS | Number of active MACs | Gives the number of active MACs. This number can differ from the number of programmed MAC addresses for instance if the external I/O access is NOT enabled. |
| *M707 | *IOMACADR | TCPIP client MAC address | Gives the external I/Os TCPIP client MAC address. |

As soon as this function starts, the UltimET tries to establish the communication with WAGO modules. The monitoring *M705 gives the status of the exchange with the WAGO modules:

| M | Alias | Name | Comment |
|-------|--------|-------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| *M705 | *IOSTS | External I/O data exchange activity | Gives the external I/O data exchange activity (10: Not active, 20: Initializing, 25: I/O read cycle, 30: I/O update cycle, 40: In error) |

As soon as the communication is in data exchange mode (*M705=25), the UltimET gets the following information from the WAGO module:

- The number of analog outputs (*M690 or *NEAOUT alias)

- The number of analog inputs (*M691 or *NEAIN alias)
- The number of digital outputs (*M692 or *NEDOUT alias)
- The number of digital inputs (*M693 or *NEDIN alias)
- The description of the I/Os controller/coupler (*M694 or *IOCPLREF alias) provides the last 3 digits of the controller/coupler reference code (e.g. 841 for the TCP/IP controller 750-841)
- The description of all connected I/O modules (*M695 or *IOMODREF alias), from depth 0 up to depth 63.

These monitorings also help checking if this matches the expected configuration.

| M | Alias | Name | Comment |
|-------|-----------|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| *M690 | *NEAOUT | Number of analog outputs | Gives the number of external analog outputs accessible through UltimET |
| *M691 | *NEAIN | Number of analog inputs | Gives the number of external analog inputs accessible through UltimET |
| *M692 | *NEDOUT | Number of digital outputs | Gives the number of external digital outputs accessible through UltimET |
| *M693 | *NEDIN | Number of digital inputs | Gives the number of external digital inputs accessible through UltimET |
| *M694 | *IOCPLREF | Converted controller/coupler reference | Gives the description of I/O controller/coupler |
| *M695 | *IOMODREF | IO modules reference | Description of the connected I/O modules, from depth 0 (first module after the coupler) up to depth 63 (last possible module) |

The monitoring *M695 stores the reference of all connected I/O modules, one at each depth. The information describing each module is defined by WAGO. Here is a copy extracted from this documentation. The first module of *M695 is not the coupler (*M694) but the first module after it (64 data instead of 65 as mentioned in the table below):

| Register address 0x2030 (MODBUS Address 408241, with a word count of upto 65) | | | | | | | | | | | | | | | | |
|-------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| Designation | Description of the connected I/O modules | | | | | | | | | | | | | | | |
| Access | Read modules 0 ... 64 | | | | | | | | | | | | | | | |
| Description | Length 1-65 words These 65 registers identify the controller and the first 64 modules present in a node. Each module is represented in a word. Because item numbers cannot be read out of digital modules, a code is displayed for them, as defined below: Bit position 0 -> Input module Bit position 1 -> Output module Bit position 2-7 -> not used Bit position 8-14 -> module size in bits Bit position 15 -> Designation digital module | | | | | | | | | | | | | | | |
| Examples: | | | | | | | | | | | | | | | | |
| 4 Channel Digital Input Module = 0x8401 | | | | | | | | | | | | | | | | |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| code | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| hex | 8 | | | | 4 | | | | 0 | | | | 1 | | | |
| 2 Channel Digital Output Module = 0x8202 | | | | | | | | | | | | | | | | |
| bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| code | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| hex | 8 | | | | 2 | | | | 0 | | | | 2 | | | |

For analog modules, the so called item number refers to the module reference (e.g. the item number 750-456 refers to an analog inputs modules, 2-channel, DC ± 10 V, Diff.).

This information is important for processing analog I/Os. For each I/O, three parameters need to be configured:

- The I/O resolution (number of bits)
- The two parameters for converting an analog raw data to an analog converted data:

- The offset applied to the raw data
- The gain for converting this raw data (+ offset) to a converted data

These parameters are NOT set automatically by the UltimET and must be set by the user. There is no automatic way for getting the number of bits of each analog I/O and the conversion parameters are user dedicated. Please refer to §4.11.2.6 and §4.11.2.7 for more information. The I/O configuration tool in ComET offers a user friendly way to set all these configuration parameters.

4.11.2.2 Enabling the update of the I/Os

To allow the user to control the state of the I/Os at initialization, the ***EIOSTA** command enables or disables the read/write cycle of the external I/Os (analog and digital).

| Command | Value | Comment |
|-----------------------|-------|---------------------------------------------------------------------|
| *EIOSTA=enable_status | 0 | Disable cyclic writing of the I/Os but keep on reading their states |
| | 1 | Enable cyclic reading and writing of the I/Os |

At the initialization, the update of the I/Os is disabled. Thus, in case the state of the outputs, set by K parameters that have been saved, is either unknown or does not correspond to the desired initial situation, the user can first set to 0 or whatever controlled value, the adequate K parameters, and then only enable the outputs to be updated.

Also, the application can decide at any moment, for instance in case of error, to freeze (perhaps after having reset) the outputs by disabling the I/O update cycle.

Also, if there is a communication error with the external I/O modules, the update of the I/Os is automatically disabled. It is up to the user to re-enable it once the error has been cleared and the situation deemed safe to pursue.

The monitoring ***M764** reflects the state of the cyclic reading and writing:

| M | Alias | Name | Comment |
|-------|-----------|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| *M764 | *ISEIOSTA | Cyclic I/O update state | 0 means cyclic writing of the I/Os is disabled, but the state of the I/Os is still read 1 means cyclic reading and writing of the I/Os is active |

Remark: From compiled sequences, the same command and register than the ones from the terminal are used.

4.11.2.3 Use of mask commands

The mask commands allow the user to read or modify more than one I/O at the same time. The mask is defined by two or three parameters:

- io_idx: this is the index of the first I/O to be read or modified
- io_number: it is the number of bits of the mask to be read or modified (maximum bits number for the mask is 16).
- Value: this parameter is only required for 'set' command. It gives the value that should replace the mask defined by 'io_firstidx' and 'io_number'. The LSB will correspond to the 'io_firstidx' bit of the I/O.

Example 1:

If the 'set' command is used like this:

Actual value of the output: B10101100

io_firstidx = 3

io_number = 3

Value = 3

The values which will be modified are the bold ones: B10**101**100

The new value of the output will be: B10011100

Example 2:

If the 'set' command is used like this:

Actual value of the output: B11110000

io_firstidx = 1

io_number =6

Value = B100011

The values which will be modified are the bold ones: B**1111**0000.

The new value of the output will be: B11000110

4.11.2.4 Digital inputs

The digital inputs are accessible through the monitoring ***M765** (or *EDINS0 alias).

| M | Alias | Name | Comment |
|--------------|---------|-------------------------|-----------------------------------------------------------------------------------------------------------|
| *M765 | *EDINS0 | External digital inputs | First bank of digital inputs (Inputs 0 to 15 stored in depth 0, ..., inputs 112 to 127 stored in depth 7) |

Remark: Each most significant 16-bit of each depth is filled with 0.
For the compiled sequences, refer to [§7.3.12.2](#).

Based on this monitoring, the access to a specific input is done through logical operations.

4.11.2.5 Digital outputs

The digital outputs are accessible through the parameter ***K760** (or *EDOUT0 alias).

| K | Alias | Name | Comment |
|--------------|---------|--------------------------|--------------------------------------------------------------------------------------------------------------|
| *K760 | *EDOUT0 | External digital outputs | First bank of digital outputs (outputs 0 to 15 stored in depth 0, ..., outputs 112 to 127 stored in depth 7) |

Before the cyclic update of the I/Os has been enabled with the *EIOSTA=1 command (refer to [§4.11.2.2](#)), the actual values of the digital outputs in the WAGO registers can be obtained by reading the monitoring ***M760**. Once the *EIOSTA=1 command has been sent, *M760 reflects the value set in *K760.

| M | Alias | Name | Comment |
|--------------|----------|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| *M760 | *EDOUTS0 | External digital outputs actual state | First bank of digital outputs actual states (outputs 0 to 15 stored in depth 0, ..., outputs 112 to 127 stored in depth 7) |

Remark: For the compiled sequences, refer to [§7.3.12.2](#).

4.11.2.6 Analog inputs

The analog outputs are accessible either as a raw data with ***M745** and ***M746** monitorings (int32 monitoring) or a converted data with ***MF745** and ***MF746** monitorings (float32 monitoring). Each register has up to 8 depths and therefore 2 registers are necessary to provide 16 analog outputs. The parameters *K615 and *K616 must be set correctly before using these inputs (see below).

| M / MF | Alias | Name | Comment |
|---------------|----------|----------------------------------|-----------------------------------------------------------------------------------------------------------------|
| *M745 | *EAINRS0 | External analog inputs raw data | First bank of analog inputs, input raw data (Input 0 stored in depth 0, ..., input 7 stored in depth 7) |
| *M746 | *EAINRS1 | External analog inputs raw data | Second bank of analog inputs, input raw data (Input 8 stored in depth 0, ..., input 15 stored in depth 7) |
| *MF745 | *EAINCS0 | External analog inputs converted | First bank of analog inputs, input converted data (Input 0 stored in depth 0, ..., input 7 stored in depth 7) |
| *MF746 | *EAINCS1 | External analog inputs converted | Second bank of analog inputs, input converted data (Input 8 stored in depth 0, ..., input 15 stored in depth 7) |

Remark: For the compiled sequences, refer to [§7.3.12.2](#).

The resolution (number of bits) of each analog input must be set in registers ***K615** and ***K616** (8 depths per register).

In WAGO analog modules, the resolution is given for unsigned data. It means that a 0-10V analog input with a resolution of 12 bits will cover a range from 0 to 4095, and a +/-10V analog input with a resolution of 12 bits will cover a range from -4096 to 4095. The sign bit comes in addition to the 12 bits resolution. For both types of analog inputs, ***K615** or ***K616** must be set to 12 (bits). The number of bits set in ***K615** and ***K616** must not count the sign bit.

The conversion must also be set. It is based on parameters ***K625** and ***K626** for the offsets and on ***KF625** and ***KF626** for the gains.

The following formula provides the conversion from raw data to converted data:

$$\text{convertedAnalogInput} = (\text{rawAnalogInput} + \text{offsetAnalogInput}) * \text{gainAnalogInput}$$

For example, the conversion of the 10th analog input is:

$$*EA\text{INC1:1} = (*EA\text{INR1:1} + *EA\text{INOFF1:1}) * (*EA\text{INGAIN1:1})$$

| K / KF | Alias | Name | Comment |
|---------------|------------|---------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| *K615 | *EAINNBIT0 | External analog inputs number of bits | First bank of analog inputs. Input number of bits (input 0 stored in depth 0, ..., input 7 stored in depth 7) |
| *K616 | *EAINNBIT1 | External analog inputs number of bits | Second bank of analog inputs. Input number of bits (input 8 stored in depth 0, ..., input 15 stored in depth 7) |
| *K625 | *EAINOFF0 | External analog inputs offset | First bank of analog inputs. Input offset (input 0 stored in depth 0, ..., input 7 stored in depth 7) |
| *K626 | *EAINOFF1 | External analog inputs offset | Second bank of analog inputs. Input offset (input 8 stored in depth 0, ..., input 15 stored in depth 7) |
| *KF625 | *EAINGAIN0 | External analog inputs gain | First bank of analog inputs. Input gain (input 0 stored in depth 0, ..., input 7 stored in depth 7) |
| *KF626 | *EAINGAIN1 | External analog inputs gain | Second bank of analog inputs. Input gain (input 8 stored in depth 0, ..., input 15 stored in depth 7) |

4.11.2.7 Analog outputs

Analog outputs can be set either as a raw data with ***K740** and ***K741** parameters (int32 parameter) or a converted data with ***KF740** and ***KF741** parameters (float32 parameter). Each register has up to 8 depths and therefore 2 registers are necessary to provide the 16 analog outputs. The parameters ***K610** and ***K611** should be set correctly before using these outputs.

| K / KF | Alias | Name | Comment |
|---------------|----------|----------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| *K740 | *EAOUTR0 | External analog outputs raw data | First bank of analog outputs. Output raw data (output 0 stored in depth 0, ..., output 7 stored in depth 7) |
| *K741 | *EAOUTR1 | External analog outputs raw data | Second bank of analog outputs. Output raw data (output 8 stored in depth 0, ..., output 15 stored in depth 7) |
| *KF740 | *EAOUTC0 | External analog outputs converted data | First bank of analog outputs. Output converted data (output 0 stored in depth 0, ..., output 7 stored in depth 7) |
| *KF741 | *EAOUTC1 | External analog outputs converted data | Second bank of analog outputs. Output converted data (output 8 stored in depth 0, ..., output 15 stored in depth 7) |

Before the cyclic update of the I/Os has been enabled with the ***EIOSTA=1** command (refer to [§4.11.2.2](#)), the actual values of the analog outputs in the WAGO module can be obtained by reading the monitorings ***M740** and ***M741** for the raw value or ***MF740** and ***MF741** for the converted value. Once the ***EIOSTA=1** command has been sent, these registers reflect the values set in the corresponding K and KF parameters.

| M / MF | Alias | Name | Comment |
|--------|-----------|-----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *M740 | *EAOUTRS0 | External analog outputs raw value | First bank of analog outputs (output 0 stored in depth 0, ..., output 7 stored in depth 7) When *M764=0 (before cyclic update of I/Os): actual raw values of outputs When *M764=1 (when cyclic update of I/Os): raw values set for outputs (*K740) |
| *M741 | *EAOUTRS1 | External analog outputs raw value | Second bank of analog outputs (output 8 stored in depth 0, ..., output 15 stored in depth 7) When *M764=0 (before cyclic update of I/Os): actual raw values of outputs When *M764=1 (when cyclic update of I/Os): raw values set for outputs (*K741) |
| *MF740 | *EAOUTCS0 | External analog outputs converted value | First bank of analog outputs, output converted data (output 0 stored in depth 0, ..., output 7 stored in depth 7) When *M764=0 (before cyclic update of I/Os): actual state of the converted values of outputs When *M764=1 (when cyclic update of I/Os): converted values set for outputs (*KF740) |
| *MF741 | *EAOUTCS1 | External analog outputs converted value | Second bank of analog outputs, output converted data (output 8 stored in depth 0, ..., output 15 stored in depth 7) When *M764=0 (before cyclic update of I/Os): actual state of the converted values of outputs When *M764=1 (when cyclic update of I/Os): converted values set for outputs (*KF741) |

Remark: For the compiled sequences, refer to [§7.3.12.2](#).

The setting of analog outputs as raw data updates the converted data of these analog outputs.
The setting of analog outputs as converted data updates the raw data of these analog outputs.

The resolution (number of bits) of each analog output is defined by parameters ***K610** and ***K611** (8 depths per register).

In WAGO analog modules, the resolution is given for unsigned data. It means that a 0-10V analog output with a resolution of 12 bits will cover a range from 0 to 4095, and a +/-10V analog output with a resolution of 12 bits will cover a range from -4096 to 4095. The sign bit comes in addition to the 12 bits resolution. For both types of analog outputs, *K610 or *K611 must be set to 12 (bits). The number of bits set in *K610 and *K611 must not count the sign bit.

The conversion must also be set. It is based on parameters ***K620** and ***K621** for offsets and on ***KF620** and ***KF621** for gains.

The following formula provides the conversion from raw data to converted data:

$$\text{convertedAnalogOutput} = (\text{rawAnalogOutput} + \text{offsetAnalogOutput}) * \text{gainAnalogOutput}$$

For example, the conversion of the 10th analog output is:

$$*EAOUTC1:1 = (*EAOUTR1:1 + *EAOUTOFF1:1) * (*EAOUTGAIN1:1)$$

| K / KF | Alias | Name | Comment |
|--------|-------------|----------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| *K610 | *EAOUTNBIT0 | External analog outputs number of bits | First bank of analog outputs. Output number of bits (output 0 stored in depth 0, ..., output 7 stored in depth 7) |
| *K611 | *EAOUTNBIT1 | External analog outputs number of bits | Second bank of analog outputs. Output number of bits (output 8 stored in depth 0, ..., output 15 stored in depth 7) |
| *K620 | *EAOUTOFF0 | External analog outputs offset | First bank of analog outputs. Output offset (output 0 stored in depth 0, ..., output 7 stored in depth 7) |
| *K621 | *EAOUTOFF1 | External analog outputs offset | Second bank of analog outputs. Output offset (output 8 stored in depth 0, ..., output 15 stored in depth 7) |
| *KF620 | *EAOUTGAIN0 | External analog outputs gain | First bank of analog outputs. Output gain (output 0 stored in depth 0, ..., output 7 stored in depth 7) |
| *KF621 | *EAOUTGAIN1 | External analog outputs gain | Second bank of analog outputs. Output gain (output 8 stored in depth 0, ..., output 15 stored in depth 7) |

4.11.2.8 Use of the I/O concept

When the feature is activated and all I/Os are configured, the user will get a cyclic update of data at the rate given by the parameter *K729 (or *IOREFRESH alias). This parameter must be configured based on the needs

of the application. The default refresh time is 10ms. The minimum value that can be used depends on the load of the UltimET. The elapsed time since the last update is accessible through the monitoring ***M708** and the maximum elapsed time is stored in the monitoring ***M706**. The monitoring ***M709** contains the update cycle count (a counter is incremented at each I/O update). This counter can be reset through the ***RIC** (Reset I/O Cycle count) command.

If the UltimET does not get a reply for the external I/O modules within the value given by the formula $*K729 + 1000$ (ms) if $*K724 = 0$ or $*K729 + *K724$ otherwise, it repeats the request $*K734$ times, incrementing each time the monitoring ***M712** (or ***IONLOST** alias). After $*K734$ repetitions, an error is generated.

| M | Alias | Name | Comment |
|--------------|-------------------|-------------------------------------------|-----------------------------------------------------------------------|
| *M706 | *IOMXRETIM | Max time between two external I/O updates | Gives the maximum elapsed time between two external I/O updates in ms |
| *M708 | *IORETIM | Time since last external I/O update | Gives the elapsed time since the last external I/O update in ms |
| *M709 | *IONCYCLE | External I/O data exchange cycle count | Gives the external I/O data exchange cycle count |
| *M712 | *IONLOST | External I/O nb of lost messages | Gives the number of lost messages on UDP/IP |

The monitoring ***M709** and the ***RIC** command can be used as handshake mechanism. The user can wait for this counter to be incremented before starting a new I/O data processing and resetting the 'maximum elapsed time between updates'.

| Command | Comment |
|-------------|-----------------------------------------------------------------------|
| *RIC | Reset the I/O update cycle count given by the monitoring *M709 |

Remark: This command can be also called from the compiled sequences.

The monitoring ***M706** is not reset by this command. It can be reset through the ***EIOMAXURST** command. This monitoring is also automatically reset after an error and at the restart of the external I/O update mechanism.

| Command | Comment |
|--------------------|------------------------------------------------------------------------------------------|
| *EIOMAXURST | Reset the observed maximum time between I/O updates given by the monitoring *M706 |

Remark: This command can also be called from the compiled sequences.

4.11.2.9 Error management

When the feature is activated, different errors may occur. Refer to [§12](#) for more information about the errors. The monitoring ***M696** (or ***IOEXC** alias) and ***M697** (or ***IOEXCSRC** alias) are used to indicate exception code meanings.

| M | Alias | Name | Comment |
|--------------|------------------|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *M696 | *IOEXC | MODBUS exception register | Gives the I/O MODBUS exception registers (depth 0 contains the MODBUS fault code (register address 1020 in WAGO); depth 1 contains the fault argument (register address 1021 in WAGO)). |
| *M697 | *IOEXCSRC | MODBUS exception source | Gives the I/O MODBUS exception source. Refer to §10 to know the different values |

When such errors occur, it is necessary to re-activate the feature through the ***RST** command. This command clears all UltimET errors and restarts the external I/O update mechanism. If the external I/O update mechanism was stopped, the ***RST** command resets also the monitoring providing the maximum elapsed time between updates (***M706**).

It is also possible to reset only the communication with the external I/O by issuing the ***RCT** command. This command restarts the UDP communication with the WAGO module. It also resets the monitorings ***M706** and ***M709** but does not reset the potential errors of the UltimET.

| Command | Comment |
|---------|---------------------------------------------------------------------------------|
| *RCT | Restarts the client TCP/IP communication and resets monitorings *M706 and *M709 |

Remark: This command can be also called from compiled sequences.

4.12 Traces management

The trace management allows the user to acquire up to 4 traces on each node, each trace having up to 16384 points.

- on a single controller (non-synchronized mode)
- on several controllers (synchronized mode with an UltimET) by using a trigger on only one node. For example, different signals could be acquired on all nodes at the beginning of the movement on node 2.

When an acquisition is made in the multi-axis scope of the ComET software, the trace acquisition is managed automatically. When an acquisition is requested by the customer's application, it may be done from the PC application or the sequence. If it is managed in the PC application using the EDI libraries (refer to the 'EDI User Manual) predefined functions are available otherwise the procedure described below must be used.

4.12.1 Traces configuration

The parameters K120 to K129 are used to set the different configurations.

| K | Name | Comment |
|-------|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *K120 | Synchronization | Enables trace synchronization mode 0 = non-synchronized mode 1 = synchronized mode |
| *K121 | Register type | Selects the type of register to trace (depth 0 for trace 0; depth 1 for trace 1,...) 1 = User variable (X) 2 = Parameter (K) 3 = Monitoring (M) 8 = Look-up table (L) 13 = Common parameter (C) 33 = User variable (XF) 34 = Parameter (KF) 35 = Monitoring (MF) 45 = Common parameter (CF) 65 = User variable (XL) 66 = Parameter (KL) 67 = Monitoring (ML) 77 = Common parameter (CL) 97 = User variable (XD) 98 = Parameter (KD) 99 = Monitoring (MD) 104 = Look-up table (L) 109 = Common parameter (CD) |
| *K122 | Register number and depth | Selects the register number (bit#0-15) and its depth (bit#16-23). Depth 0 for trace 0; depth 1 for trace 1,... |
| *K123 | Sampling period | Sampling time between 2 points of the trace (period of 25us). The minimum value is 2 for AccurET and 4 for UltimET (in non-synchronized mode) |
| *K124 | Edge selection | For trigger mode, specify if the trig condition is fulfilled on falling (-1), rising (1) or both edges (0) |

| K | Name | Comment |
|-------------------------------------|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *K125 | Trigger mode | Trace trigger mode selection: 0 = Immediate acquisition 1 = Start of movement (bit moving 0=>1 (bit#20 of M63)). For UltimET, the parameter K126 is used to specify the corresponding interpolation group. 2 = End of movement (bit moving 1=>0 (bit#20 of M63)). For UltimET, the parameter K126 is used to specify the corresponding interpolation group. 3 = Trigger at a position defined by KL127 (K126: not used) 4 = Trigger at a given value (defined in K127, KF127, KL127 or KD127) of the trace defined in K126 5 = Trigger never starts. An external freeze command in synchro mode can stop the acquisition 6 = Trigger at a given value (defined in K127 or KF127, KL127, KD127) of a register defined in K126 7 = Trigger on a bit field state (K127 or KL127:0: bit to be low, K127 or KL127:1: bit to be high) of a register defined in K126 8 = Trigger on a bit field rising or falling edge (K127 or KL127:0: bit on rising edge, K127 or KL127:1: bit on falling edge) of a register defined in K126 |
| *K126 | Trigger parameter | Definition of the trigger parameter according to the trigger mode (K125) K126:0 = register type (same possible values as K121) K126:1 = register number + depth (same possible values as K122) |
| *K127 *KD127 *KF127 *KL127 | Trigger level | Definition of the trigger level according to the trigger mode (K125) |
| *K128 | Points number | Number of points to acquire for the traces (if K128 = 0, the maximum number of points is taken into account) |
| *K129 | Pre- / post-trigger | Number of points before or after the trigger. If this number is negative, a pre-trigger is defined and if this number is positive, a post-trigger is set. To get the pre- / post-trigger time, compute $K129 * K123 * 25\mu s$. |

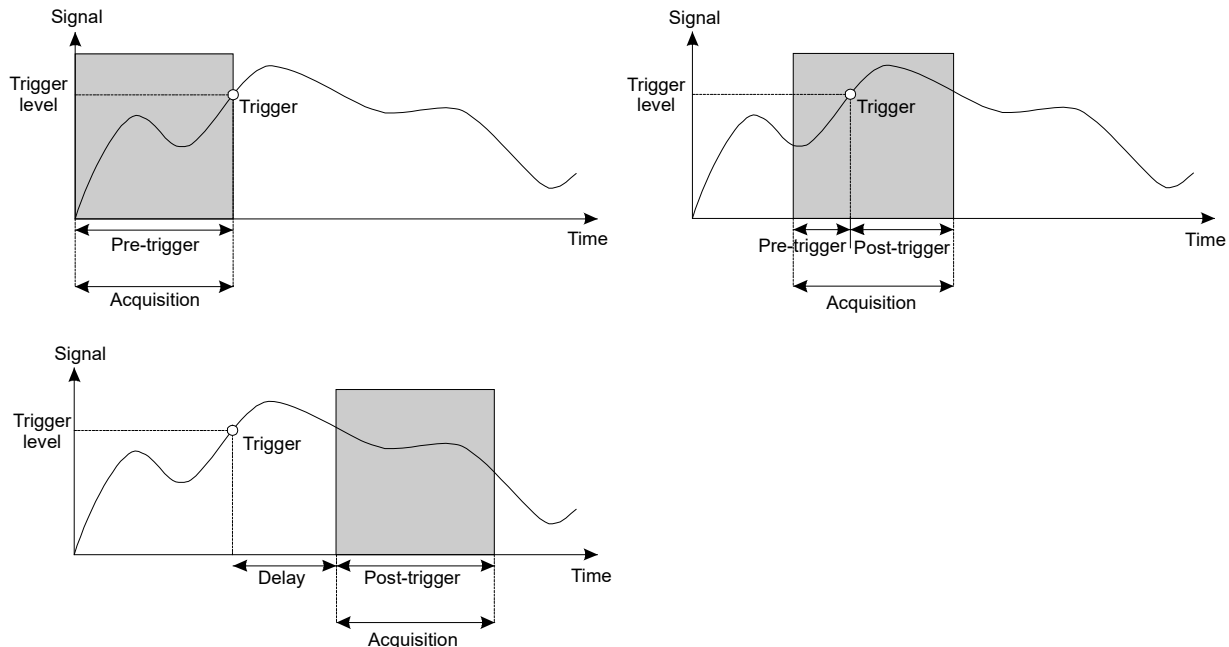
The traces configuration is stored into an internal structure and is accessible through monitoring registers. With these monitorings, the application (ComET, user application, sequence) has the information on what has been traced. Typically, ComET will be able to upload unknown traces of a controller and show which registers were traced, and when.

| M | Name | Comment |
|---------------------------|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *M120 | Synchronization mode | Gives the traces synchronization mode |
| *M121 | Register type | Gives the type of register to trace (depth 0 for trace 0; depth 1 for trace 1,...) |
| *M122 | Register number and depth | Gives the register number and its depth (bit#0-15 for register number, bit#16-23 for depth) |
| *M123 | Sampling time | Gives the sampling time between 2 points of the trace (period of 25us) |
| *M124 | Edge selection | Trace trigger edge |
| *M125 | Trigger mode | Gives the trigger mode used for the trace |
| *M126 | Trigger parameter | Gives the trigger parameter according to the trigger mode |
| *M127 *MF127 *ML127 | Trigger level | The value gives the trigger level according to the trigger mode |
| *M128 | Points number | Gives the number of acquired points for the traces |
| *M129 | Pre- / post-trigger | Gives the pre- or post-trigger value |
| *M131 | Acquisition state | Gives the acquisition state of the trace: 0 = Initialization 1 1 = Initialization 2 2 = Wait for buffer synchronization 3 = Wait for trigger 4 = Trigger condition reached 5 = End of acquisition |
| *M134 *ML134 | Freeze time | Gives the freeze time of the measure |
| *M135 | T register number at trig condition | Gives the number of the T register at the trigger condition |
| *M136 | Points number | Gives the number of available points |

4.12.1.1 Pre- and post-trigger capabilities

The user is able to visualize pre-trigger only, post-trigger only or pre-trigger and post-trigger:

- Pre-trigger: The user wants to understand the cause of an error. The error event is identified by a trigger, when the trigger occurs, the user has the capability to analyze the cause of this error by an acquisition of parameters on different traces.
- Post-trigger: The user has the possibility to analyze what is happening after trigger events on different parameters. For instance, if an error occurs, it is possible to analyze what are the consequences on other parameter. This analysis can be done from the trigger event or by choosing a delay after the trigger event.



When an acquisition with a pre-trigger is defined, the measure can stop before having all the points (defined by the parameter K128) if the trigger condition is reached before the whole pre-trigger time. To be sure that all points are available, monitorings M135 and M136 have to be checked.

4.12.2 Non-synchronized mode

The non-synchronized mode (on a single controller) is enabled when the parameter K120 = 0. To start the acquisition, the ZTE command must be used:

| Command | <P1> | Comment |
|----------------------|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *ZTE[.<axis>] = <P1> | 0 | Stops the acquisition (nevertheless the start and end index are available) Enables single axis (if already enabled: restart of the function) if K120 = 0: starts immediately the acquisition if K120 = 1: starts at the next interrupt 0 as soon as the master-of-acquisition has sent the order to start the buffering. If this order never comes, the acquisition will never happen (refer to §4.12.3) Enables multi-axes. if K120 = 0: starts immediately the acquisition if K120 = 1: the axis receiving this command is enabled and becomes the master-of-acquisition (refer to §4.12.3). |
| | 1 | |
| | 2 | |

Remark: In non-synchronized mode (K120=0), *ZTE.<axis> = 1 or *ZTE.<axis> = 2 will have the same effect as there is only one controller.

Example:

Example of an acquisition on axis 0 with 2 traces: the real position given by the monitoring ML7 and the real force given by the monitoring MF31. The acquisition has 16001 points with a sampling time of 50 μ s and lasts 800 ms. The trigger is raising edge sensitive and is set on the real position (ML7) with a level of 1000000 incr. A pretrigger of 200 ms is chosen.

```
K120.0 = 0x0; // Traces synchronization mode
K121.0 = 0x43,0x23,0x0,0x0; // Trace register type selection
```

```

K122.0 = 0x7,0x1F,0x0,0x0; // Trace register number and depth
K123.0 = 2; // Traces sampling time selection
K124.0 = 1; // Trace trigger edge selection
K125.0 = 4; // Trace trigger mode selection
K126.0 = 0,0; // Selection of the trigger parameter
K127.0 = 0,0; // Selection of the trigger level
KF127.0 = 0.0F,0.0F; // Selection of the trigger level
KL127.0 = 1000000L,0L; // Selection of the trigger level
KD127.0 = 0.0D,0.0D; // Definition of the trigger level
K128.0 = 16001; // Number of points to acquire
K129.0 = -4000; // Pre or post trigger value

```

As soon as the traces parameters are set, the command ZTE can be sent:

```
ZTE.0 = 1; // Trace trigger enable
```

When M131 = 5 or when the bit#27 of M63 is reset, the acquisition is finished and the points can be read.

4.12.3 Synchronized mode

The traces can be synchronized:

- on both axes of a dual-axis controller through USB or Ethernet
- from the master of the TransnET or from another PC connected with USB to each controller, which the trace is wanted from, but in that case, the controllers must be linked by the TransnET communication bus.

To synchronize the traces, a master-of-acquisition must be defined. This master, which must be an axis involved in the synchronized acquisition (UltimET or AccurET), will send the order (via TransnET) to start the buffering to every node (from then on, the acquisition starts on the same interrupt). To perform a measure, the user has to set first the parameters K120 to K129 of all axes. The parameter K123 as well as the parameter K128 must be identical in each axis (if an UltimET is present, the minimum value for K123 is 4).

Then, the ZTE = 1 command (refer to [§4.12.2](#)) must be sent to all axes, except to the master-of-acquisition, where the command is ZTE = 2. When the acquisition time (taking into account the fulfil of the trig condition, the acquisition time and the pre, post trigger value) is reached in an axis, this node stops all the measures.

On UltimET, the reference time counter is given by the monitoring **M228** or **ML228**.

| M | Name | Comment |
|-----------------|------------------------|------------------------------------------------------------------------------------------------------------------------------|
| *M228 *ML228 | Reference time counter | Reference time counter in unit of 25us. This counter runs freely and is synchronized with UltimET when TransnET is connected |

Example:

Example of a synchronized acquisition between axis 0 and UltimET:

On axis 0, two traces are measured, the real position given by the monitoring ML7 and the real force given by the monitoring MF31. The reference time counter given by the monitoring M228 is measured. The acquisition has 10001 points with a sampling time of 100us (the minimum possible with UltimET) and lasts 1000ms. The trigger is raising edge sensitive and is set on the real position (ML7) with a level of 1000000incr. A pretrigger of 200ms is chosen.

Parameters of axis 0:

```

K120.0 = 0x1; // Traces synchronization mode
K121.0 = 0x43,0x23,0x0,0x0; // Trace register type selection
K122.0 = 0x7,0x1F,0x0,0x0; // Trace register number and depth
K123.0 = 4; // Traces sampling time selection
K124.0 = 1; // Trace trigger edge selection
K125.0 = 4; // Trace trigger mode selection
K126.0 = 0,0; // Selection of the trigger parameter
K127.0 = 0,0; // Selection of the trigger level
KF127.0 = 0.0F,0.0F; // Selection of the trigger level
KL127.0 = 1000000L,0L; // Selection of the trigger level
KD127.0 = 0.0D,0.0D; // Definition of the trigger level
K128.0 = 10001; // Number of points to acquire
K129.0 = -2000; // Pre or post trigger value

```

Parameters of UltimET:

```

*K120 = 0x1; // Traces synchronization mode
*K121 = 0x0,0x0,0x3,0x0; // Trace register type selection
*K122 = 0x0,0x0,0xE4,0x0; // Trace register number and depth

```



```

*K123 = 4;           // Traces sampling time selection
*K124 = 0;           // Trace trigger edge selection
*K125 = 5;           // Trace trigger mode selection
*K126 = 0,0;         // Selection of the trigger parameter
*K127 = 0,0;         // Selection of the trigger level
*KF127 = 0.0F,0.0F;  // Selection of the trigger level
*KL127 = 0L,0L;      // Selection of the trigger level
*KD127 = 0.0D,0.0D;  // Selection of the trigger level
*K128 = 10001;       // Number of points to acquire
*K129 = 0;           // Pre or post trigger value

```

As soon as the traces parameters are set, the ZTE commands can be sent. The UltimET is chosen as the master-of-acquisition, thus the command ZTE is sent first to axis 0:

```
ZTE.0 = 1;           // Trace trigger enable
```

Finally, the command to start the acquisition on all nodes is sent to the UltimET:

```
*ZTE = 2;           // Trace trigger enable
```

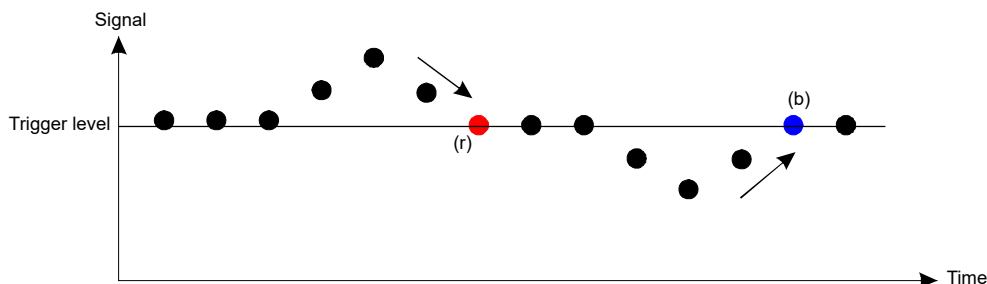
When M131 = 5 or when the bit#27 of M63 is reset, the acquisition is finished and the points can be read.

4.12.4 Trigger condition test and edge detection

The trigger condition is tested at each minimum sampling period (50us for AccurET and 100us for UltimET). It means that if an event fitting the trigger condition occurs and disappears during a sampling period (parameter K123), it will be detected but not necessarily visible on the trace.

For edge sensitive trigger conditions, the following rule is applied: the trigger condition is fulfilled when the signal reaches or crosses the trigger level in the direction of the defined edge (either falling edge or rising edge or both edges).

The following example illustrates the trigger condition detection: the falling edge detection occurs on the red (r) dot and the rising edge detection occurs on the blue (b) dot.



4.12.5 Traces upload

Each node receiving the ZTE = 1 or 2 command, sets its 'trace busy flag' and bit#27 of M63 until it finishes its acquisition or until it gets a ZTE = 0 command. When this bit is reset, the values of the traces can be uploaded.

To upload the traces, it is necessary to know the type and format of what was acquired. This information is given by the monitoring M121 describing the traces' type. Depending of the type, the user has to upload each point using T, TL, TF or TD or with the functions available with EDI libraries. There is no translation if the type is not respected, and if the user uploads a trace in double with T, the return values will have no meaning.

For example, the upload of the first trace of the acquisition set in [§4.12.2](#) can be done by asking the value TL0.0, TL1.0, TL2.0, ..., TL16000.0 to the controller.

4.12.6 Continuous traces

The aim of the "continuous traces" is to continuously upload data from an endless acquisition. This feature can be part of the process in order to get regular and continuous measurements, or can be used for debugging. It is mandatory to use the dedicated EDI functions for this special mode. Please read the related EDI documentation for more information.

It is to note that a special Wait command is available for the synchronization of the data upload: WTD (Wait Traces Data). This wait command will return as soon as all acquisition points are available.

| Command | <P1> | <P2> | Comment |
|------------------|-------------------------------|------------------------------|-----------------------------------------------------------------|
| *WTD = <P1>,<P2> | First index of the data block | Last index of the data block | Wait the end of the acquisition of the points from <P1> to <P2> |

In addition, a warning (M66 = 30) is available in case of data corruption. It will be raised when the pointer of the last written point overtakes the last uploaded point. This warning will be cleared as soon as a new acquisition is started (ZTE = 1 or 2) or after a RSD command.

4.13 Errors and warnings management

There are two different ways of managing the errors:

- by the UltimET with the parameters *KL630, *K631 and *KL632
- by the TransnET with the parameters K140 and K141

The method by 'TransnET' is recommended.

4.13.1 Errors propagation by the UltimET

This chapter describes how the UltimET Light can detect errors and how it reacts. When an error occurs, the action of the UltimET Light can be programmed. The parameter *K631 allows the user to choose between different behavior.

| K | Value | Comment |
|-------|-------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *K631 | 0 | - Stops the interpolated movements, sends the emergency command !ERR.<KL632> and jumps of the threads to the error routine |
| | 1 | - Stops the interpolated movements, sends the emergency command !HLO.<KL632> and jumps of the threads to the error routine |
| | 2 | - Brakes the interpolated movements with a programmed deceleration, sends the emergency command !HLB.<KL632> and jumps of the threads to the error routine |

The parameter *KL632 defines the mask of the nodes which the action is applied to.

| K | Comment |
|--------|--------------------------------------------------------------------|
| *KL632 | In case of error, mask of the nodes which the action is applied to |

The following paragraphs describe more precisely the different actions.

4.13.1.1 Sending of an emergency command

In case of error, the UltimET Light is able to send an emergency command to some nodes on the TransnET. The purpose is to be able to stop and cut the power in the required nodes without waiting. The commands available at the moment are:

- !ERR.<*KL632>: sends the ERR command to all the nodes defined by the parameter *KL632. This command triggers an error in the controller which opens the power bridge and displays external error.
- !HLO.<*KL632>: sends the HLO command to all the nodes defined by the parameter *KL632. Thus, the controller opens the power bridge but does not display any error.
- !HLB.<*KL632>: sends the HLB command to all the nodes defined by the parameter *KL632 and UltimET Light. The deceleration of the axes belonging to the interpolation group is given by the UltimET Light parameters corresponding to the movement types. The ones of the axes which do not belong to the interpolation group, is given by the value of their KL206 parameter. The controller does not display any error.

Refer to the '**AccurET Operation & Software Manual**' for more information about those commands. To know how to jump to an error routine, refer to [§7.6](#)

4.13.1.2 Nodes error monitoring

The UltimET Light is able to automatically monitor the error status of all the nodes present on the TransnET. When an error occurs on one of these nodes, the UltimET Light detects it and reacts.

The parameter ***KL630** allows the user to define which nodes the UltimET Light has to monitor. Each one of the 63 bits of the parameter ***KL630** represents a node. If the bit is equal to 1, the node is monitored otherwise it is not.

| K | Comment | | | |
|---------------|-------------------------------------------------------|-------|--------------------------------|--------------------------------|
| *KL630 | Mask of the nodes to monitor for node error detection | | | |
| Bit 63 | Bit 62 | | Bit 1 | Bit 0 |
| 0 | Enables node 62 error detection | | Enables node 1 error detection | Enables node 0 error detection |

When one or several nodes defined by the parameter ***KL630** are in error, the UltimET Light also generates the error 4200. As the UltimET Light is in error, the calling process of the error routine and the sending of an emergency command (described above) normally takes place.

4.13.2 ERR command

From firmware 2.07A, it is possible to add an optional parameter to the command ***ERR [= <P1>]**. It allows the user to set application errors through the sequence for example. In that case, **<P1> must be a negative value.**

- If **<P1>** is a negative value, ***M64=<P1>**, the error message in ***M95** is 'EXTERNAL ERROR' and the message on ComET application is 'ERROR <P1>: unknown'.
- For **<P1>** with positive value or the command without parameter **<P1>**, ***M64=116**, the error message in ***M95** is 'EXTERNAL ERROR' and the message on ComET application is: 'ERROR 116': This error is generated by the ERR command'.

| Command format | Comment |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *ERR [= <P1>] | Puts the controller in error mode (M64=116). If <PAR1> is a negative value, the error code in *M64 is not 116 but the value of <PAR1> defined by the user. |

4.13.3 Errors propagation on the TransnET

On all AccurETs and UltimET, there is a mechanism of errors propagation running through the TransnET. It brings two main advantages compared to the use of the UltimET's parameters ***KL630**, ***K631** and ***KL632**.

- Possibility to define 4 independent error groups of axes
- Safer because the error information runs at a lower software level

This mechanism is managed by two parameters:

- The parameter ***K141** allows the user to define the group in which the error must be published when an error occurs on the axis:

| K | Name | Comment |
|--------------|----------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| *K141 | Error published group mask | Gives the error propagation published group mask: bit#0 for group 1, bit#1 for group 2, bit#2 for group 3 and bit#3 for group 4. |

The monitoring **M141** allows the user to check the value of the error propagation register on the TransnET.

| M | Name | Comment |
|--------------|------------------------------|--------------------------------------------------------------------|
| *M141 | TransnET error groups status | Gives the value of the error propagation register on the TransnET. |

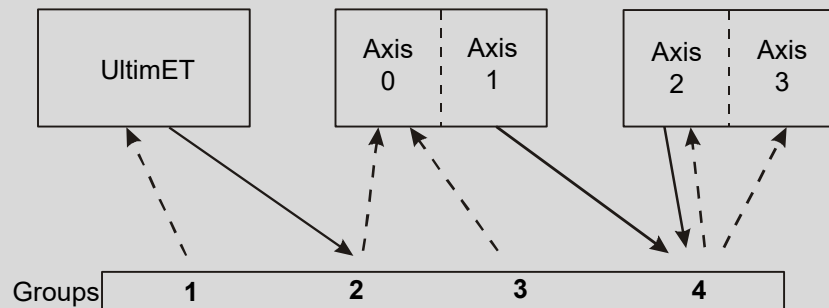
- The parameter *K140 allows the user to define which error group the UltimET belongs to. When the bit corresponding to its group(s) is set to 1 on the TransnET, the UltimET goes immediately in '**Management error, other node in error**' error (M64=4200):

| K | Name | Comment |
|-------|--------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| *K140 | Error checked group mask | Gives the error propagation checked group mask: bit#0 for group 1, bit#1 for group 2, bit#2 for group 3 and bit#3 for group 4. |

Remark: On the UltimET side, both modes are available: *KL630 and *KL632 or *K140 and *K141 but the one with the parameters *K140 and *K141 is recommended.

It is not possible to power on an axis as long as one device belonging to the same error group is in error. Now a RSD command will create an error on all devices. This command must be followed by the *RST.!=123 when the stage protection is activated. It is not possible to reset only one axis of an error group and perform on it a power on. To do this power on, all axes of the error group must be reset.

4.13.3.1 Example of use



The setting corresponding to the above-mentioned drawing is as follows:
Errors put into groups 2 to 4

```
*K141 = 0x2;           // UltimET signals its error in group 2
K141.0 = 0x0;          // Axis 0 does not signal its error in any group
K141.1 = 0x8;          // Axis 1 signals its error in group 4
K141.2 = 0x8;          // Axis 2 signals its error in group 4
K141.3 = 0x0;          // Axis 3 does not signal its error in any group
```

Errors generated from groups 1 to 4

```
*K140 = 0x1;           // UltimET must enter in error when group 1 has an error
K140.0 = 0x6;          // Axis 0 must enter in error when groups 2 or 3 have an error
K140.1 = 0x0;          // Axis 1 does not watch any group
K140.2 = 0x8;          // Axis 2 must enter in error when group 4 has an error
K140.3 = 0x8;          // Axis 3 must enter in error when group 4 has an error
```

4.13.4 TransnET functioning in case of error

4.13.4.1 Isolated communication error

One communication error occurring sometimes is normal on a 1Gbit/s communication bus like the TransnET. The effect on the system is:

- Normal or emergency command, or monitoring: the command is normally sent and possibly repeated if the TransnET error continues. The process is totally invisible for the users of the UltimET Light.
- Status: values of the last valid status are stored in the UltimET Light and transmitted to the PC as long as the error persists.
- This error is not indicated neither in the error bit of the status nor in *M64.

The monitoring ***M629** allows the user to know the number of error occurred on the TransnET.

| M | Comment |
|-------|------------------------------|
| *M629 | TransnET frame error counter |

4.13.4.2 Fatal communication error

If the frequency of the isolated error is too high, the UltimET Light enters error mode. The effect on the system is:

- The error bit of the UltimET Light's status is activated.
- *M64 stores the error 1000: '**Communication error. TransnET error**' (if it is the first one).
- All the nodes become 'not present' (present bit and all other bits of the node status at 0) except the UltimET Light.

4.13.5 Errors reset

The RST command enables the user to reset the UltimET Light and nodes' errors. If the UltimET Light or a node stays in error mode after this command, it means that the error cannot be reset. Thus, the cause has to be eliminated. For example, the error 1000 ('Communication error. TransnET error') cannot be reset as long as the TransnET does not work properly.

Syntax:

***RST[.<axis>]**

Example:

```
*RST;           // reset the errors of the UltimET Light
*RST.!;         // reset all the nodes and UltimET Light's errors
```

Due to the AccurET stage protection functions, the reset command, RST.!=123 may be sent. The command *RST.!=123 can also be executed without any further impact on the UltimET.

4.13.6 Warnings

Warnings are used to draw the attention of the user to some specific non-critical condition or to alert them when some limit is about to be reached. They are enabled/disabled with parameter ***K166** and observed by monitorings ***M166** and ***M66**. Several warnings can be signaled (active) at the same time. Each one is represented by its own bit in *K166, *M166 and a warning code in *M66.

Parameter ***K166** is a bit field allowing the users to mask (or not) each warning message individually. The default value for this parameter is 0x0 and means no warning is masked. When a bit of *K166 is set, the corresponding warning is masked and will not appear in *M66 and in bit#23 of *M63.

| K | Name | Comment |
|--------------|--------------|-----------------------------------------------------------|
| *K166 | Warning mask | Gives the bits masking the corresponding warning messages |

Example:

If the user wants to mask the «Breakpoint» warning, *K166 = 0x80000.

Monitoring ***M166** is a bit field allowing the users to see all signaled warnings. When a warning is present, the bit related to this warning is set to 1.

| M | Name | Comment |
|--------------|--------------|------------------------------------------------------------------|
| *M166 | Warning flag | Bit field where each set bit indicates an active warning message |

Each warning has a fixed relative priority. The bit number of the signaled non-masked warning having highest priority is shown by monitoring ***M66** (or **WARCD** alias).

| M | Name | Comment |
|-------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| *M66 | Warning code | Gives the bit number of the highest-priority warning that is active and enabled in *K166 (refer to §11 for the complete list) |

The presence of any signaled non-masked warning is indicated by setting bit #23 (warning) in *M63. ComET

shows the presence of an enabled non-masked warning by putting an yellow circle on top of the 'UltimET' icon in its status bar. If there is simultaneously an error signaled, it is the red circle for the error that has the priority. The bit definitions of *M166 and *K166 are identical.

4.14 Status

4.14.1 UltimET Light controller status

The 'controller status', given by the monitoring **M63**, allows the user to see if the controller is in error, is executing a movement, is running a sequence, etc. This monitoring variable is a 32-bit field with the following meaning:

| Bit # | Value | Name | Description (when bit = 1) |
|----------|------------|--------------|--------------------------------------------------------------------------------|
| 0 to 15 | - | - | User status bit defined by parameter K177 (refer to §10. for more information) |
| 16 to 18 | - | - | Reserved |
| 19 | 524288 | Present | 1 = node present; 0 = node not present |
| 20 | 1048576 | Moving | The UltimET Light is executing an interpolated trajectory (bit#25 OR bit#29) |
| 21 to 22 | - | - | Reserved |
| 23 | 8388608 | Warning | There is an active warning |
| 24 | 16777216 | Sequence | The UltimET Light is executing an internal sequence |
| 25 | 33554432 | Ipol0 moving | The UltimET Light is executing an interpolated trajectory, group 0 |
| 26 | 67108864 | Error | 1 = The UltimET Light is in error mode, 0 = OK |
| 27 | 134217728 | Trace | Trace busy flag is set during a register trace acquisition |
| 28 | - | - | Reserved |
| 29 | 536870912 | Ipol1 moving | The UltimET Light is executing an interpolated trajectory, group 1 |
| 30 | 1073741824 | - | The TransnET connection has been interrupted |
| 31 | 2147483648 | - | The controller is executing thread depending setting *K297 |

The content of this register is updated each time a change occurs. At each cycle, slave nodes (AccurET) transmit 32 status bits to the UltimET Light. These 32 bits enable the user to have real-time information concerning the nodes. These information are a copy of the monitoring M63 of each nodes. These status are available in the monitoring ***M520** and each of its depth corresponds to a node.

| Monitoring | Comment |
|--------------|----------------------------------------------------------------------------------------------------------------|
| *M520 | Copy of the monitoring M63 of all the nodes present on the TransnET. The depths correspond to the node number. |

Remark: Refer to §10. for more information about the status given by the monitoring M63.

The users status (bit 0 to 15) are a copy of *K177 parameter of each node. It is a fast process to transmit information from the slaves to the master. A value written by an UltimET Light in its *K177 parameter will be available at the next cycle in the UltimET Light. The trajectory generator can also directly write in K177 (refer also to §6.6) to indicate a few points along the trajectory.

4.14.2 'Sequence' bit of the UltimET Light

The 'sequence' bit of the UltimET Light indicates if one of the UltimET Light's thread is working. Only one of the thread has to work to have the bit set to 1. To know which threads are active, the monitoring ***M97** has to be consulted. Each bit of this monitoring represents a thread (refer to §7. for more information).

| Monitoring | Comment |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *M97 | Mask of the sequence threads currently active (bit1: 1 = thread 1 on, 0 = thread 1 off, bit2: 1 = thread 2 on, 0 = thread 2 off, bit3: 1 = thread 3 on, 0 = thread 3 off) |

4.14.3 'Interrupt' bit

Each time the status is modified on a node, it generates an interruption at the PCI, PCIe or TCP/IP bus level of the PC for the UltimET Light. This functionality is used by the libraries available for the UltimET Light in order to detect a few events as the end of a movement. Without that, the PC should all the time question the UltimET Light and will waste its calculation time.

Associated to the user status, these interruptions are a powerful tool which enables an UltimET Light or AccurET's sequence to inform the PC of all events interesting for the user.

4.15 Set / reset bit(s) management

The **CH_BIT_REG32** command allows the user to perform a reset / set on some bits in a register (integer register K, C and X) without affecting the other bits of this register.

| Command format | Comment |
|----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *CH_BIT_REG32.<axis> = <P1>,<P2>,<P3> | Write any bit (set bits and/or clear bits mask) <P2> with defined mask value <P3> of any 32 integer register <P1> without modifying the other bits the register |

This will do $\langle P1 \rangle = (\langle P1 \rangle \& \sim \langle P2 \rangle) | (\langle P3 \rangle \& \langle P2 \rangle)$

Example:

Thread 1 controls bits 0, 1 and 4 of K177:0.<axis>.

Thread 1 wants to set bit#0 and 4, and clear bit#1, without modifying all the other bits. The command will be:
CH_BIT_REG32.<axis>=K177:0.<axis>, 0x13, 0x11

Thread 1 wants to clear bit#4 and set bit#0 and 1, without modifying all the other bits. The command will be:
CH_BIT_REG32.<axis>=K177:0.<axis>, 0x13, 0x3

Thread 2 wants to clear bit#2 and set bit#3, without modifying all the other bits. The command will be:
CH_BIT_REG32.<axis>=K177:0.<axis>, 0xC, 0x4

As the CH_BIT_REG32 command cannot be interrupted, there will be no conflict between threads.

4.16 Set several registers

The **SSR** command (Set Several Registers) allows the user to assign up to 6 registers in the same command.

| Command format | Comment |
|--------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *SSR[.<axis>] = <P1>,<P2>,[<P3>,...<P12>] | Command to set several registers: <P1>: 1st register and depth definition, <P2>: 1st register value, <P3>: 2nd register and depth definition (optional parameter), <P4>: 2nd register value (optional parameter),..., <P11>: 6th register and depth definition (optional parameter), <P12>: 6th register value (optional parameter). |

This command is only available for X, XL and XF registers. In the same command, it is allowed to set different types of registers for example:

SSR[.<axis>] = X1:1, 100, XF1:1, 24.523F, XL3:1, 1234L

If the register number is greater than the maximum allowed for the type and if the register type is different than X, XL and XF, the '**ERROR PARAM**' error (M64=3000) will occur.

It is possible to use the indirect parameterization on the SSR command but the Y value (indirect parameterization) is the same for all the registers in the command: SSR[.<axis>] = XY:1, 10, XY:2, 20, XFY:1, 10F.

5. Real-time channels on TransnET

5.1 Principle

In some application it is necessary to transmit data between the ETEL devices. There are two ways to share the data between the AccurET, the UltimET and the PC:

- by sending commands from an UltimET sequence or in a PC application, that reads a controller register and writes its value in a register of another controller.
- by continuously sharing data through the TransnET with specific commands introduced in this chapter.

This continuous data sharing, also called Real-Time Values (RTV), can be done between:

- the UltimET and AccurET controllers
- the controllers themselves
- the application PC and the controllers (with UltimET PCI / PCIe only).

The TransnET bus works with time cycles of 100µs. Each 100µs, a TransnET frame starts from the UltimET (TransnET bus master) and is shared between the different connected controllers. A part of the TransnET frame is made up of slots (a slot is 32-bit data word) used for cyclic data transfer at each TransnET cycle.

The following information may be present on this cyclic data range:

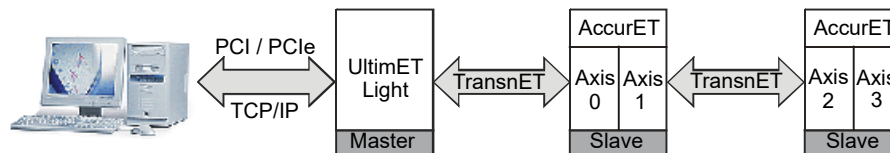
- Axes status information (given by the monitoring M63)
- Monitoring requests (AccurET to UltimET, 4 slots per axis)
- User defined Real-Time Values (RTV)
- ...

There are two types of RTV:

- RTV0: they are managed by the UltimET and described in this chapter (refer to [§5.3.1](#)).
- RTV1: for future use

The number of RTV0 slots depends on the number of present axes: 298 slots – (4 x number of axes (for monitoring request)) - (2 x number of interpolated axes).

Example:



The number of RTV0 is:

- Without interpolation: $298 - (4 \times 4) = 282$
- With interpolation on axes 2 and 3: $298 - (4 \times 4) - (2 \times 2) = 278$

5.2 Data sharing

5.2.1 Between controllers

The read or write on these cyclic data slots is done through sharing data functions (refer to [§5.3.2](#)). The user has to request UltimET for free RTV0 slots before using these data sharing functions. This kind of functions allows one controller to have access to another controller registers in real time.

The EDI interface and sequences provide access to all these functions. The request of RTV0 slots has to be done either on the PC application or from an UltimET sequence. Refer to the '**EDI User's Manual**' for more information about the complete list of the DSA functions and example describing how to use RTV using EDI functions

5.2.2 Between controllers and the PC application (UltimET Light PCI / PCIe only)

When using real-time operating systems (for those where EDI is supported or without EDI), this is possible to have access to the TransnET cyclic data slots at the PCI / PCIe interface.

EDI offers a set of functions allowing the sharing of real-time cyclic data with the controllers. Refer to the '**EDI User's Manual**' for more information about the complete list of the DSA functions and example describing how

to use RTV using EDI functions. A user defined handler can be called at each TransnET cycle (or multiples) to execute cyclic tasks. Watchdog functionality is provided to warn the user if the interrupt acknowledge cannot be processed at the correct frequency (the time of the handler is not taken into account).

5.2.3 Disappearing of TransnET

Each time a TransnET connection is established, the configuration of the TransnET slots is initialized. Depending on the number of axes, the RTV0 range will be automatically updated. This also means that each time the TransnET disappears and appears again, the assignment of slots is cleared in the UltimET and in the controllers. The configuration of the cyclic data slots has to be redone by the application and/or in the UltimET sequence.

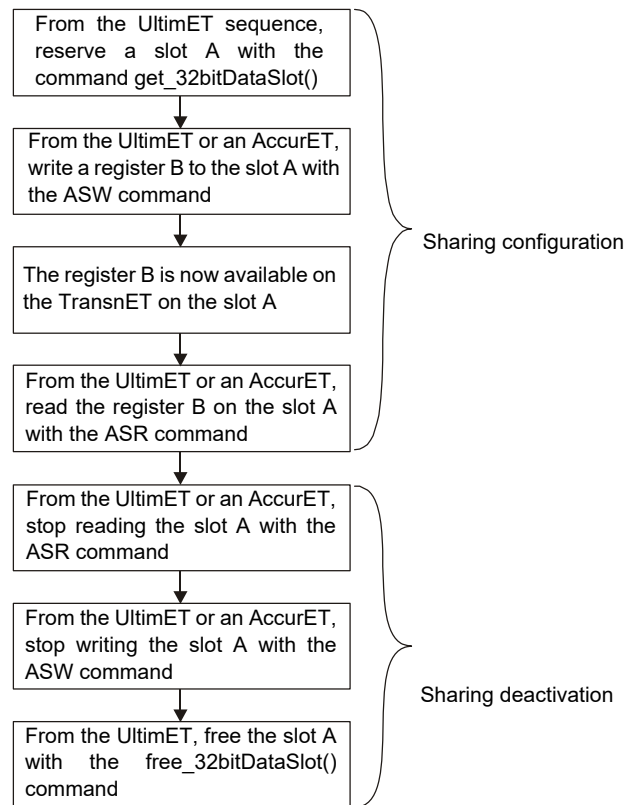
To provide an event to the user, bit#30 of the monitoring *M63 is set as soon as the TransnET connection is interrupted. This status bit is cleared when the TransnET connection is established and when the UltimET reset error command (*RST) has been executed. It is thus necessary to check this event at the application level, and/or in the sequence of the UltimET.

5.3 Interface description

The interface description is presented in three parts:

- Commands related to the management of RTV0 slots
- Commands related to the sharing of the data on the TransnET
- A special UltimET command allowing any command to be forwarded to a mask of axes

Here is a summary on the data sharing principle:



The quantity of slots to be shared in 32-bit integer is:

| | Read from TransnET | Write on TransnET |
|---------|------------------------------------|------------------------------------|
| AccurET | 8 or 16 | 7 or 15 (*) |
| UltimET | 16 | 8 |
| PC | All available slots (PC dependant) | All available slots (PC dependant) |

(*): 1 slot is already used for sharing the monitoring M63 (controller status) of each controller.

5.3.1 Slot management

The following UltimET sequence commands are used to manage the slots. These commands cannot be used in the 'Terminal' tool of ComET.

- `int get_32bitDataSlot()`
 - get a free TransnET slot
 - no parameter needed
 - returns the value of the slot
 - returns the value -1 if no more slot is available
- `void free_32bitDataSlot(int slot_number)`
 - free the mentioned slot number

The following UltimET monitoring are used to manage the slots:

| M | Name | Comment |
|-------|---------------------------|-------------------------------------------|
| *M524 | Number of free RTV0 slots | Gives the number of free RTV0 slots. |
| *M525 | First RTV0 slot number | Gives the number of the first RTV0 slot. |
| *M526 | Number of RTV0 slots | Gives the number of available RTV0 slots. |

Each time the 'get_32bitDataSlot()' command is executed, a new slot is allocated. The following example illustrates what may happen if slots are not properly managed and how to restore the initial situation.

Example:

in this example, the system is made of an UltimET and one AccurET controller (axes 0 and 1). The initial number of RTV0 slots is: $*M524 = 298 - (2 \times 4) = 290$ free RTV0 slots.

```
int slot_axis0; // Variable that will store the allocated slot
                // number

slot_axis0 = get_32bitDataSlot(); // *M524 is now equal to 289
slot_axis0 = get_32bitDataSlot(); // Error: this second request of slot
                                // allocation must not be done using the same
                                // variable. *M524 is now equal to 288
```

At this state, the first slot number previously stored in `slot_axis0` is no more available.

```
free_32bitDataSlot(slot_axis0); // One slot, whose value is stored in
                                // slot_axis0, is free. *M524 is now equal to
                                // 289
```

The first requested slot cannot be free with the 'free_32bitDataSlot' command. There are two possibilities to free all the RTV0 slots:

- Making the TransnET disappeared, either by powering off/on the UltimET or by sending a RSD.!=255 command

- Freeing all slots, even those that are not allocated. The following sequence loop does this task:

```
for (i=*M525;i<(*M525+*M526);i++) // Free all slots from the first RTV0 slot
                                //(*M525) to the last (*M525+*M526-1)
{
    free_32bitDataSlot(i);
}
```

5.3.2 UltimET – AccurET functions for sharing data

Sharing data means the exchange of data between controllers and the access of TransnET cyclic data. To do so, the *ASR and *ASW commands (in urgent and normal record formats) are used. With these commands, the request of a free RTV0 slot to the UltimET is mandatory.

5.3.2.1 ASW (ASsign for Write) command

Syntax:

***ASW[.<axis>]=register_to_share, slot_lsl [,slot_msl]**

| Parameter | Comment |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| register_to_share | Defines the register to be written by the controller (RTV) in the TransnET cyclic data. If this parameter = 0, the register is not written any more in the TransnET cyclic data |
| slot_lsl | Defines the slot in the TransnET cyclic data where the controller writes the register, lsl part (RTV) |
| [slot_msl] | Defines the slot in the TransnET cyclic data where the controller writes the register, msl part (RTV). Only available for 64-bit registers |

To stop this continuous write, the `*ASW=0,slot_lsl [,slot_msl]` command must be sent.

Remark: 'lsl' means 'least significant long' and 'msl' means 'most significant long'.

The number of slots already used by the `*ASW` command is given by the monitoring M449. There are 8 slots reserved for this function for the UltimET. This maximum number of slots for `*ASW` command is available in monitoring M443. Each slot number associated with an ASW is accessible in `*M448`.

| M | Name | Comment |
|--------------|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *M443 | Maximum number of RTV for ASW | Gives the maximum number of RTV slot available for the ASW command. |
| *M448 | Used RTV slots (by ASW) | Gives the slot number associated to a register (the depth corresponds to each of the possible reserved slots). |
| *M449 | Counter of written RTV | Gives the number of written RTV by <code>*ASW</code> command. By default the monitoring M63 is already put on RTV and can be read through UltimET (<code>*M520:axis_number</code>) |

If the `*ASW` command does not have the right format (too many/few arguments, slot number out of range,...), the following errors can occur: '**BAD_SLOT_TRANSNET**' (M64=58) for AccurET and '**ERROR_SLOT_ASW_BAD_PARAM**' (M64=1210), '**ERROR_SLOT_NO_FREE_ASW_SLOTS**' (M64=1211) '**ERROR_SLOT_ALREADY_IN_USE**' (M64=1202) for UltimET.

The status of all axes is automatically updated in the UltimET by using cyclic data slots. The nodes statuses (given by the monitoring M63 of each axis) are available through the monitoring `*M520`.

| M | Name | Comment |
|--------------|----------------------------|----------------------------------------------------------------------------------------------------------------|
| *M520 | Copy of the monitoring M63 | Copy of the monitoring M63 of all the nodes present on the TransnET. The depths correspond to the node number. |

Remark: One slot of each controller is used for transmitting the status information to the UltimET. It means that the number of available slots is reduced by one in each axis for the ASW command. The monitoring M449 is thus by default equal to 1.

5.3.2.2 ASR (ASsign for Read) command

Syntax:

`*ASR[.<axis>]=monitoring_register, slot_lsl [,slot_msl]`

| Parameter | Comment |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| monitoring_register | Defines the monitoring register that is read as a RTV register in the TransnET cyclic data: M450 to M465, or MF450 to MF465, or ML450 to ML457 or MD450 to MD457 |
| slot_lsl | Defines the slot number where the data or the lsl part of the data is stored in TransnET. |
| [slot_msl] | Defines the slot number where, only for 64-bit data, the msl part is stored in TransnET. |

There 16 slots reserved for the `*ASR` command for the UltimET Light. This maximum number of slots is available in monitoring M447.

| M | Name | Comment |
|--------------|-------------------------------|---------------------------------------------------------------------|
| *M447 | Maximum number of RTV for ASR | Gives the maximum number of RTV slot available for the ASR command. |

Remark: Refer to §5.3.2.1 to have an example.
It is not possible to use at the same time M450 for a slot and MF450 for another slot (same memory range used).

| Offset of memory range | M or MF | ML or MD |
|------------------------|---------|-----------|
| 0 | 450 | 450 (msl) |
| 1 | 451 | 450 (lsl) |
| 2 | 452 | 451 (msl) |
| 3 | 453 | 451 (lsl) |
| 4 | 454 | 452 (msl) |
| 5 | 455 | 452 (lsl) |
| 6 | 456 | 453 (msl) |
| 7 | 457 | 453 (lsl) |
| 8 | 458 | 454 (msl) |
| 9 | 459 | 454 (lsl) |
| 10 | 460 | 455 (msl) |
| 11 | 461 | 455 (lsl) |
| 12 | 462 | 456 (msl) |
| 13 | 463 | 456 (lsl) |
| 14 | 464 | 457 (msl) |
| 15 | 465 | 457 (lsl) |

To stop reading a register, the following command must be sent: ASR=monitoring_register, -1, [-1]. If a register reading is not necessary, this command is used to save process time of the PLTI interrupt of the AccurET or the high priority interrupt in UltimET.

If the *ASR command does not have the right format (too many/less arguments, slot number out of range,...), the following errors can occur: **'BAD_SLOT_TRANSNET'** (M64=58) for AccurET and **'ERROR_SLOT_ASR_BAD_PARAM'** (M64=1220) for UltimET.

*ASW and *ASR commands example:

Sharing and unsharing the real position of the axis 0 to axes 1, 2 and 3 and UltimET. Such a sequence must be used in the UltimET controller only.

// Sharing and unsharing of a 64-bit position from axis 0 to axes 1, 2 and UltimET.
// This sequence must be executed in UltimET only.

```
int axis0_real_position_slot_lsl; // RTV slot for lower 32-bit of position ML1
int axis0_real_position_slot_msl; // RTV slot for upper 32-bit of position ML1

// RTV start
void func10(void)
{
    // Get RTV slots
    axis0_real_position_slot_lsl = get_32bitDataSlot();
    axis0_real_position_slot_msl = get_32bitDataSlot();
    // Axis 0 writes its real position ML1 on TransnET
    ASW.0 = ML1, axis0_real_position_slot_lsl, axis0_real_position_slot_msl;
    // Axes 1, 2 and UltimET start to get ML1 of axis 0 through their own ML453
    *ASR = *ML453, axis0_real_position_slot_lsl, axis0_real_position_slot_msl;
    ASR.(1,2) = ML453.!, axis0_real_position_slot_lsl, axis0_real_position_slot_msl;
}

// RTV stop
void func20(void)
{
    // Axes 1, 2 and UltimET stop reading the 2 RTV slots and free their own ML453
    *ASR = *ML453, -1, -1;
    ASR.(1,2) = ML453.!, -1, -1;
    // Axis 0 stops writing its real position ML1 on TransnET
    ASW.0 = 0, axis0_real_position_slot_lsl, axis0_real_position_slot_msl;
    // Free RTV slots
```

```

    free_32bitDataSlot(axis0_real_position_slot_lsl);
    free_32bitDataSlot(axis0_real_position_slot_msl);
}

```

5.3.2.3 SETRTV command

An alternative to the *ASW command is the *SETRTV command allowing the user to directly change a value in an RTV. This feature is helpful to quickly start a pending action on AccurETs.

Syntax:

***SETRTV=slot, value**

| Parameter | Comment |
|-----------|-------------------------------------------------------------------------|
| slot | Defines the slot in the TransNET cyclic data where the value is written |
| value | Defines the value (INT32) written in the slot |

Example:

Several AccurET controllers execute a sequence which waits for a change in their M450 monitoring (pre-set using *ASR command) before starting their movement.

```

void func100(void)
{
    WPG.0 = M450.0,1;
    // Move start: move start type is chosen by the customer
    // E.g.
    MVE.0 = XL100.0;
    ...
}

```

From the UltimET, the RTV slots have been reserved and the ASR commands of the controllers have been executed.

```

slotStartMove = get_32bitDataSlot(); // Ask a free slot
ASR.(1,2,3) = M450.T,slotStartMove;
...
*SETRTV = slotStartMove,0;           // Write 0 on the slot
JMP.(1,2,3) = 100;                   // AccurET sequence waits for M450 to
                                     // be at 1 before starting their
                                     // movement.
...
*SETRTV = slotStartMove,1;           // Write 1 on the slot

```

In AccurET controllers sequence, condition is true at the next AccurET cycle. Then all movements start. Before restarting this AccurET sequence function, it is necessary to send the *SETRTV = slotStartMove, 0 command.

5.3.3 Real-time channels timing

Here are the different delays and update rates:

| Configuration | Delay | Update rate |
|---------------------------------------|---------------------|-------------|
| AccurET <=> AccurET | 150 µs (worst case) | 100 µs |
| AccurET <=> UltimET | 200 µs (worst case) | 100 µs |
| AccurET (via UltimET PCI/PCle) <=> PC | PC dependant | 100 µs |
| UltimET (PCI/PCle) <=> PC | PC dependant | 100 µs |

5.3.4 Forward command

The ***FWD (ForWarD)** command of the UltimET allows the user to send any command to AccurET controllers by giving the axes destination mask as a parameter. This command can be forwarded as an urgent or normal record. This is mainly for sequence use, because it is not possible to specify the axes mask as a variable in standard commands.

Syntax:

***FWD = dest_axes_mask, cmd_number, is_urgent_cmd [, <P1>, <P2>, ..., <P6>]**

| Parameter | Comment |
|-----------------------|--------------------------------------------------------------|
| dest_axes_mask | Gives the destination axes mask in long format [hexadecimal] |
| cmd_number | Gives the command number in int format |
| is_urgent_cmd | Normal command record (0) or urgent command record (1) |
| [<P1>] | First parameter of the command |
| [<P2>] | Second parameter of the command |
| [<P2>] | Third parameter of the command |
| [<P4>] | Fourth parameter of the command |
| [<P5>] | Fifth parameter of the command |
| [<P6>] | Sixth parameter of the command |

If the forwarded command returns with an error, the *FWD commands also returns with the '**Command error. Command's parameters not correct**' error (M64=3000).

Example:

Sending of a 'power on' on axes 0 and 1.

```
*FWD = 0x3L,124,0,1; // 0x3L is the axes mask
                      // 124 is the command number for the PWR command
                      // 0 indicates that the command is sent as a normal command
                      // record (this command does not exist in the urgent format)
                      // 1 indicates that the axes are in power on (PWR.(0,1) = 1).
```

THIS PAGE IS INTENTIONALLY LEFT BLANK

Chapter D: Interpolation commands

6. Interpolation

6.1 Introduction

The interpolation commands are available only in the UltimET version with interpolation (refer to [§2.4](#) for the ordering information).

It is important to understand the processing of the interpolation commands (in the UltimET Light) to know the functioning of these commands. There are different kinds of interpolation commands:

- the ones used exclusively to manage the interpolation groups (setting, activation of a group,...)
- the ones which contribute to the movement of the axes or give its progress (line, arc of a circle, marks,...)
- the ones which directly interfere on the current movement (stop, cam on the speed,...)

The first and third kind of commands are run as soon as they are processed by the command manager of the UltimET Light. The commands relating to the movement are pre-processed at the command manager level and then stored in a buffer (refer to [§2.3.3](#) for more information about the buffer).

Remark: All the interpolation commands have a parameter to specify the interpolation group. This parameter is equal to 0 or 1 corresponding to the first or second interpolation group respectively. To differentiate the commands going to the interpolation buffer from the other commands, refer to the UltimET Light commands list in [§8](#).

6.1.1 Units of the interpolation commands

At the application interface, interpolation commands parameters can be defined either in ISO or in power of ten (10^x) of ISO unit. The ISO units are:

- Position: in meter for linear motor and in turn for rotary motor
- Speed: in meter/second for linear motor and in turn/second for rotary motor
- Acceleration: in meter/second² for linear motor and in turn/second² for rotary motor
- Time: in second

The power of ten can be defined by the user and saved in K parameters (*K522, *K523, *K524, *K525, *K526). This choice has not only an effect on the meaning of the command's parameters but also on the precision of the trajectory calculated by the UltimET Light. These units have a direct influence on the internal resolution of the UltimET Light trajectory calculator. The position has an additional parameter (*K522 parameter) to define a multiplying factor of the power of ten. It enables the user to work, for example, with a unit of half a micrometer for the position.

Two depths are possible for these parameters: depth 0 for the interpolation group 0 and depth 1 for the interpolation group 1. Currently only the first depth of these parameters is used by EDI, meaning that the conversion factors in EDI are based on the values defined for interpolation group 0, even for group 1.

| Parameter | Description |
|-----------|---------------------------------------------------------------------------------------|
| *K522 | Multiplying factor for the position. The depths correspond to the ipol group numbers. |
| *K523 | Power of ten for the position. The depths correspond to the ipol group numbers. |
| *K524 | Power of ten for the speed. The depths correspond to the ipol group numbers. |
| *K525 | Power of ten for the acceleration. The depths correspond to the ipol group numbers. |
| *K526 | Power of ten for the time. The depths correspond to the ipol group numbers. |

The units are as follows:

- Unit for position in interpolated command = $*K522 \times 10^{*K523}$ [meter or turn] = ufpi (user friendly position increment)
- Unit for speed in interpolated command = 10^{*K524} [meter / second or turn / second] = ufsi (user friendly speed increment)
- Unit for acceleration in interpolated command = 10^{*K525} [meter / second² or turn / second²] = ufai (user friendly acceleration increment)
- Unit for time in interpolated command = 10^{*K526} [second] = ufti (user friendly time increment)

Example:

To work in a linear system with the following units for the interpolation group 0:

- position in tenth of micrometer
- speed in micrometer/second
- acceleration in micrometer/second²
- time in millisecond

The K parameters have to contain the values:

```
*K522:0 = 1;      // Multiplies the position by 1
*K523:0 = -7;     // Power of ten for the position, -7 means 0.1 micrometer
*K524:0 = -6;     // Power of ten for the speed, -6 means 1 micrometer / second
*K525:0 = -6;     // Power of ten for the acceleration, -6 means 1 micrometer
                  // second2
*K526:0 = -3;     // Power of ten for the time, -3 means 1 millisecond
```

Remark: In compiled sequences, when ISO units are used, a conversion of units is realized and done at compilation time and not at run-time. Thus, parameters *K522, *K523.... must be defined in the UltimET before starting the sequence compilation. Changing these parameters at run-time will not be properly taken into account and must be avoided for sequences when interpolation commands parameters are defined in ISO.

6.1.2 Distance and time limits for interpolation commands

Most of the interpolation commands need parameters representing a distance. All these distances have to be in the limits described in this chapter.

The limit is the same for all the axes of the interpolation group. The maximum distance, given by the monitoring ***M659**, results from the definition of the position unit in the UltimET Light (*K522 and *K523 parameters), the controllers resolution and the maximum value of a 32 bits signed integer. The controller resolution used in the formula is the one of the interpolated axes which has the highest resolution. The maximum distance is limited either by the controller resolution or the position unit definition. **The maximum length of a segment is limited to this value: Max_length = controller resolution [m] x (2³²-1).**

The most restrictive value gives the maximum distance.

Max_distance [m] (= *M659) = minimum (controller_resolution [m] x (2³¹-1); *K522 x 10^{*K523} x (2³¹-1)).

The minimum distance is:

Min_distance = - max_distance

The maximum time value, given by the monitoring ***M658**, results from the definition of the time unit in the UltimET (*K526 parameters).

Max_time [s] (= *M658) = 10^{*K526} x (2³¹-1).

| Monitoring | Comment | Unit |
|--------------|---------------------------------------------------------------------------------------------|------|
| *M658 | Maximal time value for interpolation commands (depth 0 for group 0 and depth 1 for group 1) | ufti |
| *M659 | Maximal distance for interpolation commands (depth 0 for group 0 and depth 1 for group 1) | ufpi |

Example 1:

An XY system forms the interpolation group 0 and has the following characteristics:

- Axis X: node 0; the resolution of the controller is 256 increments per micrometer.
- Axis Y: node 1; the resolution of the controller is 1024 increments per micrometer.

The Y axis has the highest resolution. Therefore, the Y axis defines the position limits of the interpolation command. If the users want to work with a position unit of one micrometer (*K522:0 = 1, *K523:0 = -6), the limits are:

$$\begin{aligned}
 \text{Max_distance} &= \text{minimum} (\text{controller resolution} \times (2^{31}-1); *K522 \times 10^{*K523} \times (2^{31}-1)) \\
 &= \text{minimum} (1 \times 10^{-6} / 1024 \times (2^{31}-1); 1 \times 10^{-6} \times (2^{31}-1)) \\
 &= \text{minimum} (2.097 ; 2147) \\
 &= 2.097 \text{ m} \\
 \text{Min_distance} &= - \text{max_distance} = -2.097 \text{ m}
 \end{aligned}$$

Example 2:

For a single axis with a low resolution of 256 increments per millimeter and a position unit of one micrometer (*K522:0 = 1, *K523:0 = -6), the position limits are:

$$\begin{aligned} \text{Max_distance} &= \text{minimum} (\text{controller resolution} \times (2^{31}-1); *K522 \times 10^{*K523} \times (2^{31}-1)) \\ &= \text{minimum} (1 \times 10^{-3} / 256 \times (2^{31}-1); 1 \times 10^{-6} \times (2^{31}-1)) \\ &= \text{minimum} \left(\frac{1}{8388}; \frac{1}{2147} \right) \\ &= 2147 \text{ m} \\ \text{Min_distance} &= -\text{max_distance} = -2147 \text{ m} \end{aligned}$$

6.2 Setting of the interpolation

This chapter describes the commands which enable the definition, activation and deactivation of a group of interpolation:

- The **ISSET** command defines the link between an interpolated axis and the associated nodes. A group of interpolation is defined by its four interpolated axes (X, Y, Z and θ).
- The **IBEGIN** command activates the interpolation group.
- The **IEND** command deactivates the interpolation group.

6.2.1 ISSET command

The **ISSET** command allows the user to define the nodes associated to the axes of the interpolation group.

Syntax:

***ISSET = ipol_group, axis_x, axis_y, axis_z, axis_θ**

| Parameter | Value | Comment |
|------------|------------------------|-----------------------------------------------------------------------------------|
| ipol_group | 0 or 1 | Interpolated axes group |
| axis_x | From 0 to $(2^{64}-1)$ | Mask of the nodes associated to X interpolated axis (integer 64.bit value) |
| axis_y | From 0 to $(2^{64}-1)$ | Mask of the nodes associated to Y interpolated axis (integer 64.bit value) |
| axis_z | From 0 to $(2^{64}-1)$ | Mask of the nodes associated to Z interpolated axis (integer 64.bit value) |
| axis_θ | From 0 to $(2^{64}-1)$ | Mask of the nodes associated to θ interpolated axis (integer 64.bit value) |

The user has 2 groups of 4 interpolated axes. To make interpolated movements, the user must determine which nodes will execute these movements and which interpolated axes they are associated to. It leads the user to determine a reference frame. These nodes associated to the interpolated axes belong to the group of interpolation determined by the ipol_group parameter.

The three axes (axis_x, axis_y, axis_z) define normally a three-dimensional orthonormal and Cartesian reference frame. The 'axis_θ' is generally considered as a rotary axis, but the use of each interpolated axis is determined by the user.

The four interpolated axes can be used for linear interpolated movements, however, only two axes (axis_x, axis_y) can be used for a circular interpolation. Circular movements, out of plane, are feasible by using the matrix functions (refer to [§6.7](#) for more information).

Example:

A system with three linear and orthogonal axes is configured as written below:

The node 0 represents the axis_x, the nodes 3 and 8 represent the axis_y and the node 12 represents the axis_z. The line of command is:

```
*ISSET = 0,1L,264L,4096L,0L; // The interpolated group number is the group 0
// The mask for the axis_x is 20
// The mask for the axis_y is 23 + 28
// The mask for the axis_z is 212
// The mask for the axis_θ is 0
```

- Remark:** An interpolated axis may contain several nodes but a node can belong to only one interpolated axis.
If an interpolated axis contains several nodes, they must be connected to the axes having a certain number of identical parameters:
- They all must be associated with either rotary or linear motors.
 - They all must be associated to the same type of encoder with the same encoders parameters.
 - The interpolation of the encoder signal must also be identical (refer to the operation & Software Manual for more information about these parameters).

A copy of these parameters is automatically stored in the M and ML monitorings (on the UltimET Light) as well as in the mask of the nodes associated to the axes. A monitoring M also defines if the interpolated axis is activated or not. Necessary information are stored in the M and ML monitorings on several levels.

The monitorings configuration is:

| Monitoring | Comment |
|------------|----------------------------------------------------------------------------------------------------------------------|
| *ML525 | Interpolation, interpolated axis mask (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M726 | Interpolation, axis present (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M727 | Interpolation, number of axes per ipol axis (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M728 | Interpolation, axis encoder ipol factor (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M729 | Interpolation, axis K77 ipol factor (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M730 | Interpolation, axis motor type (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M731 | Interpolation, axis encoder period (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M732 | Interpolation, axis motor mult. factor (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M733 | Interpolation, axis motor div. factor (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |

The depth of each monitoring corresponds to an interpolated axis. For example, the list of the present axes given by the monitoring *ML525, is:

| | |
|-----------|------------------------------------------------|
| *ML525: 0 | Axis X of the interpolated axes group 0 |
| *ML525: 1 | Axis Y of the interpolated axes group 0 |
| *ML525: 2 | Axis Z of the interpolated axes group 0 |
| *ML525: 3 | Axis θ of the interpolated axes group 0 |
| *ML525: 4 | Axis X of the interpolated axes group 1 |
| *ML525: 5 | Axis Y of the interpolated axes group 1 |
| *ML525: 6 | Axis Z of the interpolated axes group 1 |
| *ML525: 7 | Axis θ of the interpolated axes group 1 |

All these registers are automatically uploaded when the ISET command is executed.

6.2.2 IBEGIN command

The **IBEGIN** command allows the user to switch in interpolated mode, the interpolation group specified in the parameter as well as the nodes associated with.

Syntax:

***IBEGIN = ipol_group**

| Parameter | Value | Comment |
|------------|--------|-------------------------|
| ipol_group | 0 or 1 | Interpolated axes group |

This command initializes the interpolation group and sends a command to all interpolated controllers to activate their interpolation flag. After this command, the nodes are in an interpolated mode and receive their position set points from the real-time channels of the UltimET Light.

The monitoring ***M530** allows the user to know at any time if the interpolated mode is active or not. The depths correspond to the ipol group numbers.

| Monitoring | Value | Comment |
|--------------|-------|-------------------------------------|
| *M530 | 0 | The interpolated mode is not active |
| | 1 | The interpolated mode is active |

The jerk time (defined in the controllers by K213 parameter) and the cam value (defined in the controllers by KF205 parameter) of the controllers are no longer active when they are in interpolated mode.

Remark: It is not possible to activate the interpolated mode when the UltimET Light is in error. First of all, find the reason of the error, correct and reset it (with the ***RST** command).

6.2.2.1 Origin point of the interpolation mode

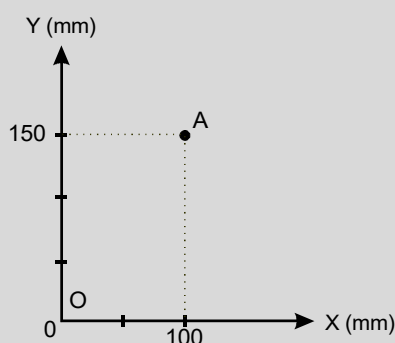
In a multi-axis system, each axis has a specific zero position. A standard reference frame (also called 'Machine reference frame') is defined by the axes orientation, and the origin of that reference frame corresponds to the point where each axis theoretical position is equal to 0. In the interpolated mode, the origin point and the reference frame may be different than the one previously defined. To do so, 3 different working modes, one relative mode and 2 absolute modes, have been developed to set the user reference frame.

- **Relative mode**

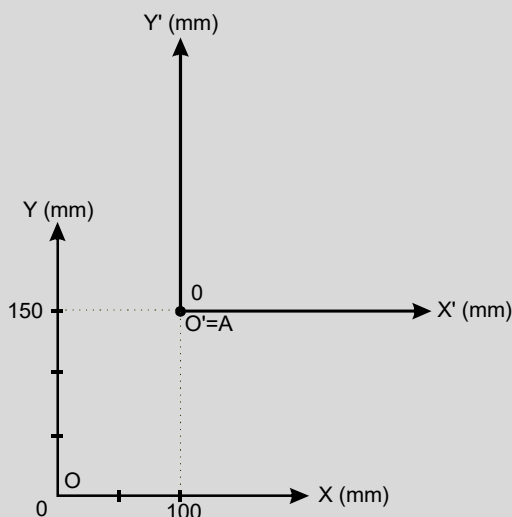
This mode is the mode by default (it also corresponds to the mode used with the DSAMX product family). The origin point of the interpolation mode reference frame (also called 'User reference frame') is defined at the execution of the IBEGIN command. Before this command, each axis has a specific theoretical position. All these theoretical positions define a point in the machine reference frame. After the execution of the IBEGIN command, that point becomes the origin point of the reference frame. As the origin point of both reference frames may be different, there should have an offset between them.

Example:

In a XY system, the machine reference frame is OXY. The theoretical positions of both axes define the point A (X=100mm, Y=150 mm) in the machine reference frame.



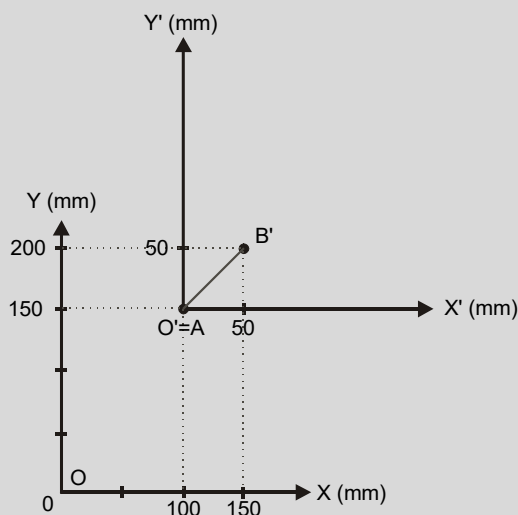
The IBEGIN command is executed at this point defining the origin point of the interpolated mode reference frame O'X'Y'. The point A in the machine reference frame is then equal to the point O' (X'=0 mm, Y'=0 mm) in the interpolated reference frame.



A linear interpolated movement is executed (user position unit in millimeter):

```
*ILINE=0,50,50,0,0; // Linear displacement to the point B' (X'=50 mm, Y'=50 mm)
```

(Refer to §6.4.1 for more information about the ILINE command).



In the machine reference frame, the coordinates of the point B' are: X=150 mm, Y=200 mm.

- **Absolute mode with the interpolation origin point defined by the origin point of each interpolated axis**

The **IABSMODE** (Interpolation **ABS**olute **MODE**) command allows the user to activate or clear the absolute reference coordinates mode of the selected interpolation group.

Syntax:

```
*IABSMODE = ipol_group, reference_mode
```

| Parameter | Value | Comment |
|----------------|--------|-----------------------------|
| ipol_group | 0 or 1 | Interpolated axes group |
| reference_mode | 0 | Clears the absolute mode |
| | 1 | Activates the absolute mode |

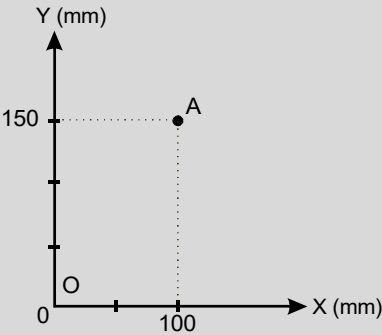
Remark: When the reference_mode is equal to 0, the mode corresponds then to the relative mode.
If there are more than one axis per interpolated axis, the first one is the one taken into account.

The monitoring ***M590** allows the user to know if the absolute mode is activated or not. The depths correspond to the interpolation group numbers

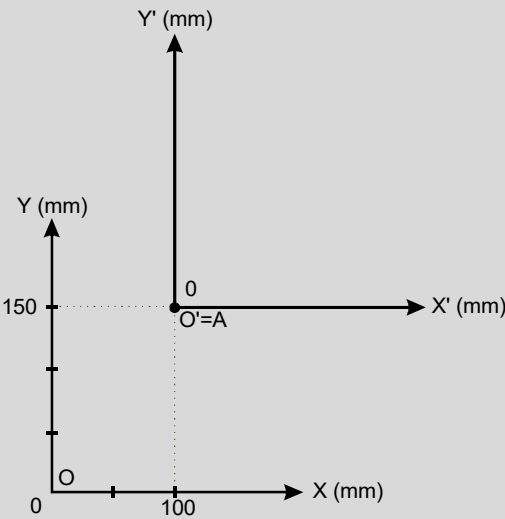
| Monitoring | Value | Comment |
|------------|-------|---------------------------------------------------------------------|
| *M590 | 0 | Absolute mode deactivated. It corresponds then to the relative mode |
| | 1 | Absolute mode activated |

Example:

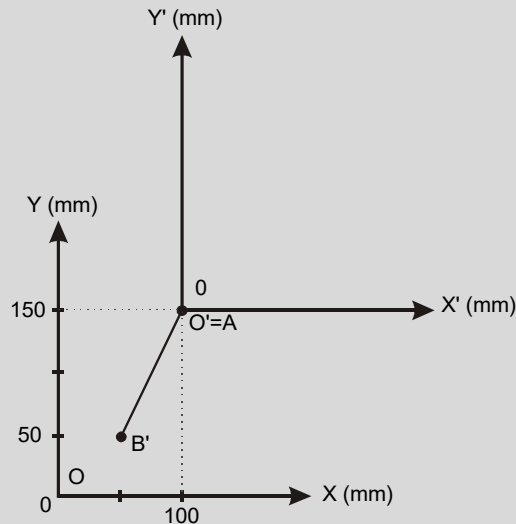
In a XY system, the machine reference frame is OXY. The theoretical positions of both axes define the point A (X=100mm, Y=150 mm) in the machine reference frame.



The IBEGIN command is executed at this point defining the origin point of the interpolated mode reference frame O'X'Y'. The point A in the machine reference frame is then equal to the point O' (X'=0 mm, Y'=0 mm) in the interpolated reference frame.



```
*IABSMODE = 0,1          // The absolute mode is activated
*ILINE = 0,50,50,0,0;    // Linear displacement from O' to point B' (X=50 mm, Y=50 mm)
```



- **Absolute mode with the interpolation origin point defined by the user**

The **IABSCOORDS** (Interpolation **ABS**olute **COORD**inate**S**) command allows the user to assign the coordinates of the current position of the axes of the selected interpolation group.

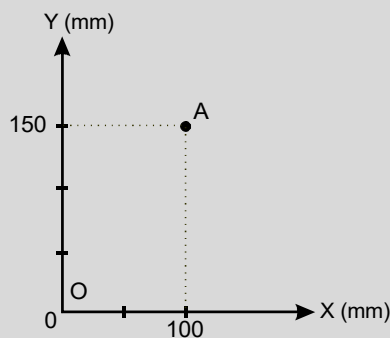
Syntax:

***IABSCOORDS = ipol_group, current_coord_x, current_coord_y, current_coord_z, current_coord_θ**

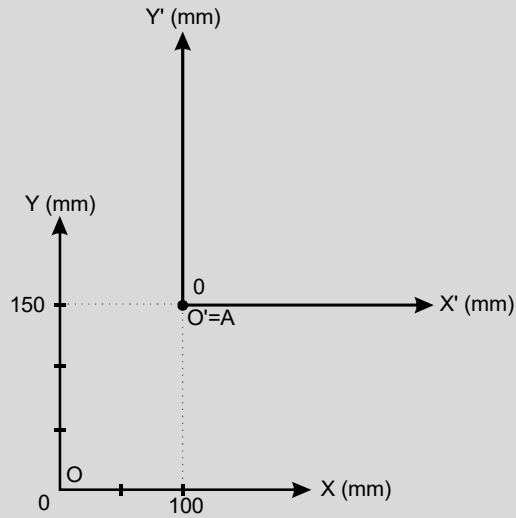
| Parameter | Value | Comment | Unit |
|-----------------|------------------------------------------|-----------------------------|----------|
| ipol_group | 0 or 1 | Interpolated axes group | |
| current_coord_x | From -distance_max to +distance_max (**) | x axis absolute coordinates | ufpi (*) |
| current_coord_y | From -distance_max to +distance_max (**) | y axis absolute coordinates | ufpi (*) |
| current_coord_z | From -distance_max to +distance_max (**) | z axis absolute coordinates | ufpi (*) |
| current_coord_θ | From -distance_max to +distance_max (**) | θ axis absolute coordinates | ufpi (*) |

Example:

In a XY system, the machine reference frame is OXY. The theoretical positions of both axes define the point A (X=100mm, Y=150 mm) in the machine reference frame.

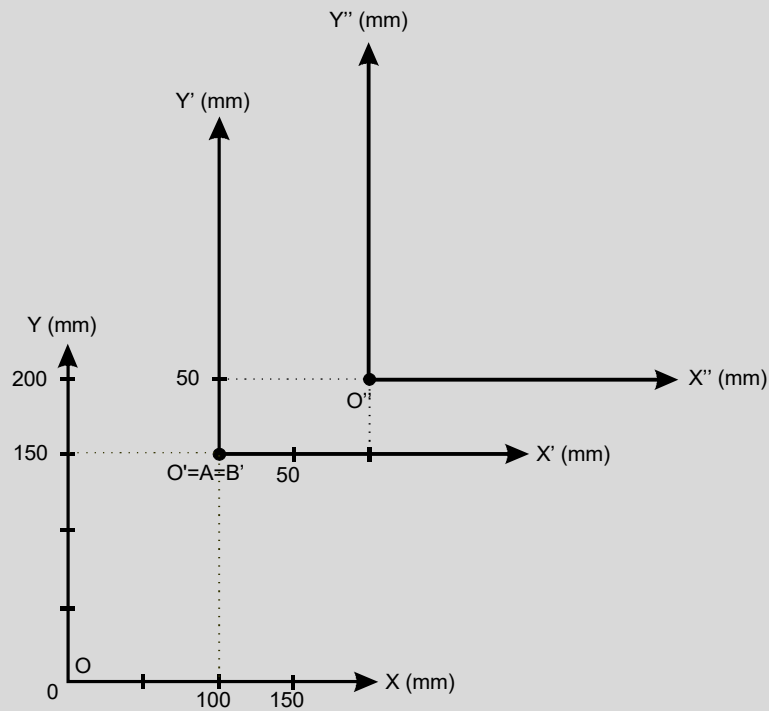


The **IBEGIN** command is executed at this point defining the origin point of the interpolated mode reference frame O'X'Y'. The point A in the machine reference frame is then equal to the point O' (X'=0 mm, Y'=0 mm) in the interpolated reference frame.



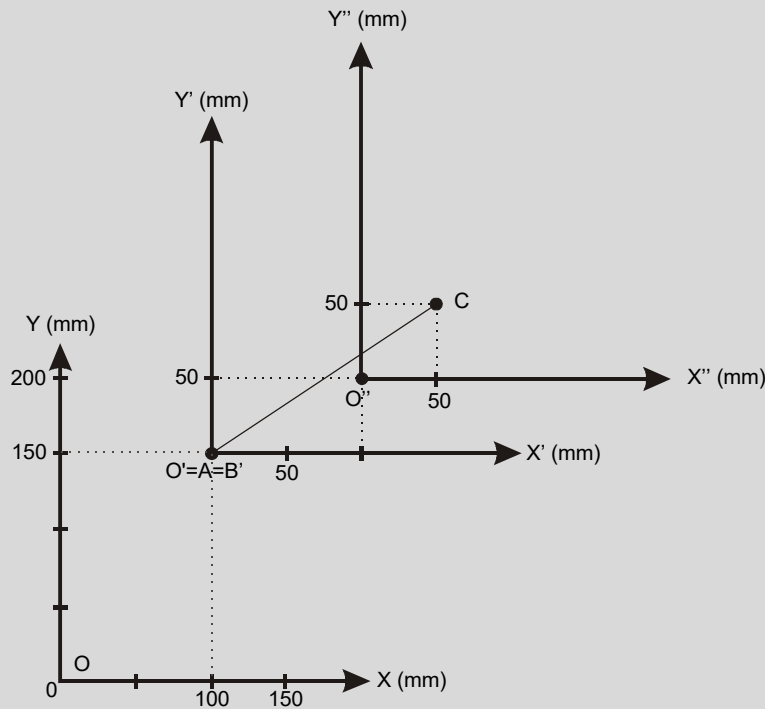
```
*IABSCOORDS = 0,-100,-50,0,0; // The current point coordinates (and then the
                                // interpolation mode reference frame) is set at
                                // point B' (X'=-100 mm, Y'=-50 mm)
```

The reference frame O''X''Y'' used for the interpolation is then as follows:



```
*ILINE = 0,50,50,0,0; // linear displacement from B' to the point C (X' '=
                       // 50 mm, Y' '=50 mm)
```

(Refer to [§6.4.1](#) for more information about the ILINE command).



6.2.3 IEND command

The **IEND** command allows the user to switch the interpolation group (specified in the `ipol_group` parameter) as well as the nodes associated with to a non-interpolated mode. This command is linked to the **IBEGIN** command.

Syntax:

***IEND = ipol_group**

| Parameter | Value | Comment |
|-------------------------|--------|-------------------------|
| <code>ipol_group</code> | 0 or 1 | Interpolated axes group |

This command deactivates the interpolation flag of the nodes involved which means that the controllers are now in normal mode and do not receive any more their position set points from the real-time channels of the UltimET Light.

As *IEND command is immediately executed, the current movement must be finished before sending this command.

6.2.4 Two groups of interpolation

When working with two interpolation groups, the parameter `*K600` defining the interpolation refresh rate has to be set to 2. This will share the processing load in two successive TransnET cycles. The reference positions will thus be sent at a rate of 200µs.

| K | Name | Comment |
|--------------|----------------------------|------------------------------------------------------------|
| *K600 | Interpolation refresh rate | Sets the interrupt refresh frequency for the interpolation |

Caution: `*K600` parameter must be set prior to the configuration of interpolation groups (`iset`, `ibegin` commands).

The activation of the interpolation groups will configure the controllers to properly handle a reference position received at this rate. If one interpolation group is started with a defined `*K600` value, it is forbidden to change it for the second interpolation group otherwise the “**Interpolation error. Error when all groups do not have the same interrupt refresh rate**” error (M64=2024) will occur.

6.3 User position vs. real position

The following monitorings allow the user to monitor the user and the real position of the axis X, Y, Z and θ of both interpolation groups.

| Monitoring | Comment |
|------------|-----------------------------------------------------------------------------------------------------------------------------|
| *M640 | User position of X axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M641 | Real position of X axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M642 | User position of Y axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M643 | Real position of Y axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M644 | User position of Z axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M645 | Real position of Z axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M646 | User position of Theta axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M647 | Real position of Theta axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M660 | User position of X axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M661 | Real position of X axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M662 | User position of Y axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M663 | Real position of Y axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M664 | User position of Z axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M665 | Real position of Z axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M666 | User position of Theta axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M667 | Real position of Theta axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |

6.4 Interpolation movements

Four different interpolation movements can be realized with the UltimET Light:

- 'G-code based' (lines/circles) with linear interpolation on 4 axes (x, y, z, θ), circular interpolation on 2 axes (CW and CCW) and control of the speed profile along the trajectory (either trapezoidal or S-curve with jerk control).
- PVT (Position-Velocity-Time, vectors) with interpolation on 4 axes (x, y, z, θ) as well as on-fly trajectory and target change.
- PT (Position-Time) with interpolation on 4 axes (x, y, z, θ).
- Universal Linear Movement (ULM) with linear interpolation on 4 axes (x, y, z, θ), automatic concatenation of segments and transitions between segments defined by time parameters.

6.4.1 'G-code based' mode

The G-code defines the coding of preliminary functions used for the numerical control of machinery. This coding is normalized at international level. The standard is ISO 6983 and called: Numerical control of machinery - Program format and definition.

This standard describes how are defined the commands of the movement generation. The four main commands are: G00 (quick axis displacement), G01 (interpolated linear displacement of the axes), G02 (interpolated displacement on an arc of a circle in the clockwise direction) and G03 (interpolated displacement on an arc of a circle in the counter-clockwise direction). The UltimET Light uses the global definition of these movements, whose command form is a compromise between the standard and ETEL language already used for the controllers.

The UltimET Light enables the interpolation of linear and circular segments on a predefined axes group. The trajectory is made up of a series of segments put end to end. According to the axes configuration, it is possible to interpolate the trajectories in a plane and / or in space.

These trajectories are defined by the type of wanted segment but also by some speed, acceleration and deceleration parameters applied along the trajectory. It is also possible to limit the jerk (derivative of the acceleration) to enable a smoother movement. A movement profile is then applied on the interpolated trajectory according to the axes limitations.

Most of the commands described in this chapter are based on G-code. They are a mix between the G-code principle and the syntax of the language (assembler) used by ETEL. ILINE, ICWR or ICW, and ICCW or ICCW commands are actually ETEL commands for G01, G02 and G03. ISET and IBEGIN commands must be executed before executing the commands written in the following chapter.

6.4.1.1 Linear interpolated movement command

The ILINE command leads to the execution of a linear interpolated movement on 4 axes.

Syntax:

***ILINE = ipol_group, coord_x, coord_y, coord_z, coord_θ**

| Parameter | Value | Comment | Units |
|------------|------------------------------------------|-----------------------------------|----------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| coord_x | From -distance_max to +distance_max (**) | x coordinates of the target point | ufpi (*) |
| coord_y | From -distance_max to +distance_max (**) | y coordinates of the target point | ufpi (*) |
| coord_z | From -distance_max to +distance_max (**) | z coordinates of the target point | ufpi (*) |
| coord_θ | From -distance_max to +distance_max (**) | θ coordinates of the target point | ufpi (*) |

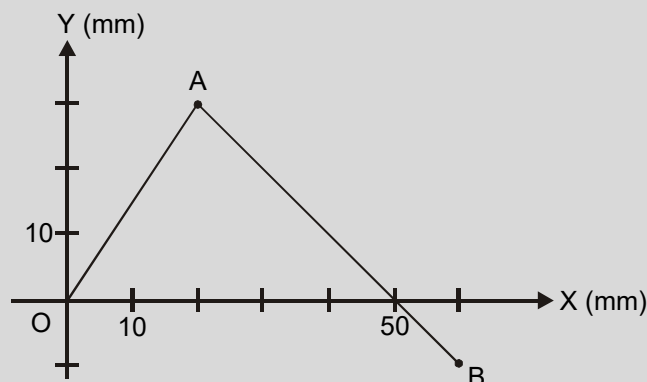
(*) and (**): Refer to [§6.1.1](#) for more information.

A movement profile is applied on this movement defined by a tangential acceleration (ITACC), a tangential deceleration (ITDEC), a tangential speed (ITSPD) and a jerk time (ITJRT). Refer to [§6.4.1.3](#) for more information.

Example:

On a system with two orthogonal and linear X and Y axes, the following segments are executed. The first segment goes from point O (X=0mm, Y=0mm) to point A (X=20mm, Y=30mm) and the second goes from point A to point B (X=60mm, Y=-10mm). The user position unit of the position is the micron. The commands are:

```
*ILINE = 0,20000,30000,0,0;
*ILINE = 0,60000,-10000,0,0;
```



6.4.1.2 Circular interpolated movement commands

Two types of commands can be used depending if the circle is defined by its center or its radius.

- **ICW and ICCW commands**

ICW and **ICCW** commands allow the user to create an arc of a circle. The arc of a circle is defined by its

direction (CW for clockwise or CCW for counter-clockwise), its starting point (current point), its end point (defined by coord_x and coord_y) and its center (defined by center_x and center_y).

Syntax:

***ICW (or *ICCW) = ipol_group, coord_x, coord_y, center_x, center_y**

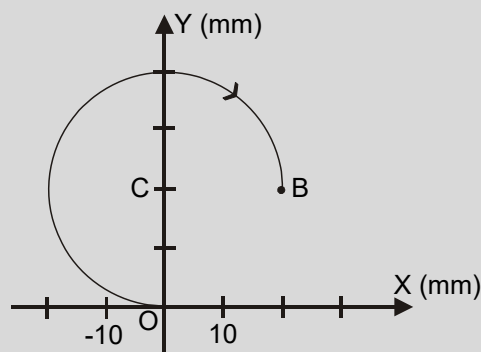
| Parameter | Value | Comment | Units |
|------------|------------------------------------------|------------------------------------|----------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| coord_x | From -distance_max to +distance_max (**) | x coordinates of the target point | ufpi (*) |
| coord_y | From -distance_max to +distance_max (**) | y coordinates of the target point | ufpi (*) |
| center_x | From -distance_max to +distance_max (**) | x coordinates of the circle center | ufpi (*) |
| center_y | From -distance_max to +distance_max (**) | y coordinates of the circle center | ufpi (*) |

(*) and (**): Refer to [§6.1.1](#) for more information.

Example:

On a system with two orthogonal and linear X and Y axes, an arc of a circle is executed in the clockwise direction, defined by a starting point O (0,0), an end point B (X=20 mm, Y=20 mm) and the center of the circle C (X= 0 mm, Y=20 mm). The user position unit is the micron. The command line is:

*ICW = 0,20000,20000,0,20000;



Wrong value in the definition of the circles

If the lengths (starting point to center of the circle) and (end point to center of the circle) are different, the circle cannot be executed in a normal way. Thus, ICW and ICCW commands introduce a correction procedure to perform the circle.

This procedure leads to the realization of a linear displacement followed by a circle whose radius is equal to the length from the end point to the center of the circle. The starting point of the circle is rectified and a line binds the bad defined starting point to the corrected one. The corrected starting point belongs to the segment defined by the wrong starting point and the center of the circle.

Example:

In a XY stage, the current point is A (X=5 mm, Y=5 mm). The interpolated mode is active and the user position unit is the micron. The following command is sent:

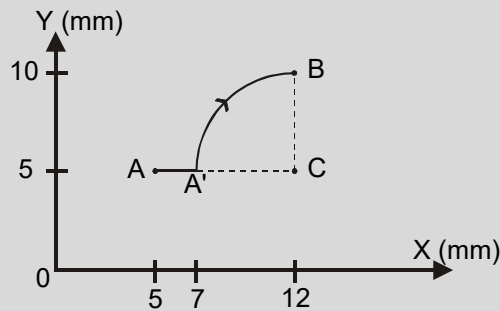
```
*ICW = 0,12000,10000,12000,5000;    // Clockwise circle to reach point B (X=12 mm,
// Y=10 mm) with a circle given by point C
// (X=12 mm, Y=5 mm)
```

This command cannot be executed because the lengths (starting point to the center of the circle) and (end point to the center of the circle) are different. The UltimET Light automatically introduces a new command line before the above-mentioned command. The executed sequence is:

```
*ILINE = 0,7000,5000,0,0;            // Linear movement to the point A' (X=7 mm,
// Y=5 mm)
*ICW = 0,12000,10000,12000,5000;    // Clockwise circle to reach point B (X=12 mm,
```

```
// Y=10mm) with a circle given by point C
// (X=12 mm, Y=5 mm)
```

The circle can be now executed properly.



• ICWR and ICCWR commands

ICWR and **ICCWR** commands allows the user to create an arc of a circle. The arc of a circle is defined by its direction (CW for clockwise or CCW for counter-clockwise), its starting point (current point), its end point (defined by coord_x and coord_y) and its radius.

Syntax:

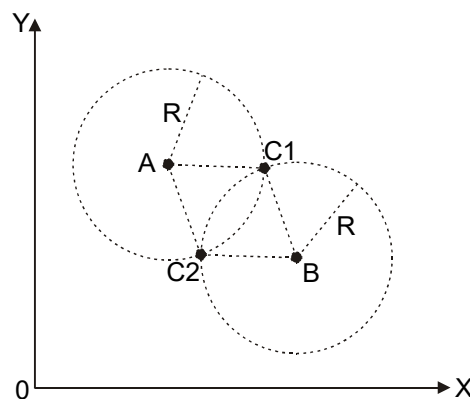
***ICWR (or *ICCWR) = ipol_group, coord_x, coord_y, radius**

| Parameter | Value | Comment | Units |
|------------|------------------------------------------|------------------------------------|----------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| coord_x | From -distance_max to +distance_max (**) | x coordinates of the target point | ufpi (*) |
| coord_y | From -distance_max to +distance_max (**) | y coordinates of the target point | ufpi (*) |
| radius | From -distance_max to +distance_max (**) | Length of the radius of the circle | ufpi (*) |

(*) and (**): Refer to [§6.1.1](#) for more information.

A movement profile is applied on this displacement. This profile is defined by a tangential acceleration (ITACC), a tangential deceleration (ITDEC), a tangential speed (ITSPD) and a jerk time (ITJRT). Refer to [§6.4.1.3](#) for more information.

During the execution of the arc of a circle with the radius as parameter, there are 2 possible centers of circle. One center will create an arc of a circle covering less than 180° and the other one more than 180°. If the radius specified in the command is positive, the UltimET Light generates an arc of a circle covering less than 180°. If the radius is negative, the UltimET Light generates an arc of a circle covering more than 180°. The following figure represents both centers of circle defined by its radius R, its starting point A and its end point B.



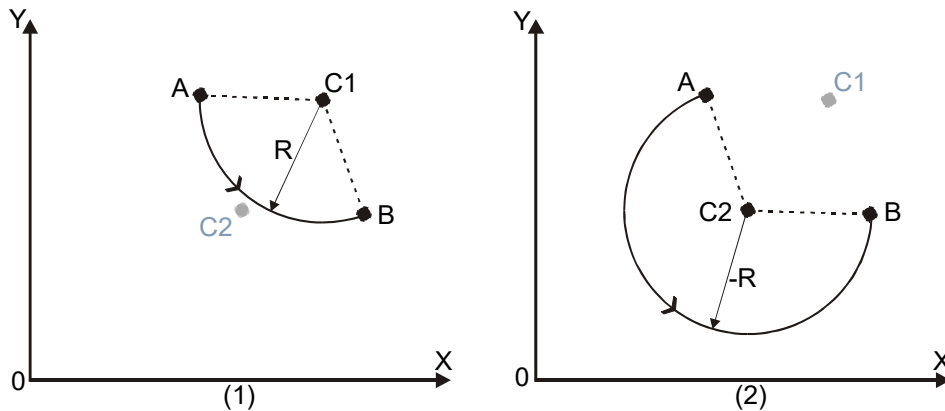
In the next figure, the arcs of a circle are covered in the counter-clockwise direction (ICCWR). If the radius R is positive, the associated center is C1. The angle formed by the points A, C1 and B is smaller than 180°. If the

radius R is negative, the associated center is $C2$. The angle formed by the points A , $C2$ and B is bigger than 180° . The commands to execute these circles in an orthogonal XY system are (with A as starting point):

*ICCW = 0, X_b , Y_b , R (1)

or

*ICCW = 0, X_b , Y_b , $-R$ (2)



With ICWR and ICCWR commands, it is not possible to complete a whole circle. One parameter is missing to define the position of the center. To execute a whole circle, ICW (clockwise) and ICCW (counter-clockwise) commands have to be used.

Wrong value in the definition of the circles

If the distance between the starting and the end point of the circle is bigger than its diameter, the circle cannot be executed in a normal way. Then, ICWR and ICCWR commands introduce a correction procedure to perform the circle. This procedure leads to the realization of a linear displacement followed by half a circle with its defined radius. The starting point of the circle is rectified and a line binds the bad defined starting point to the corrected one.

Example:

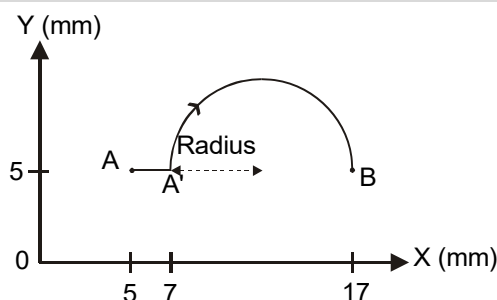
In a XY stage, the current point is A ($X=5$ mm, $Y=5$ mm). The interpolated mode is active and the user position unit is the micron. The following command is sent:

```
*ICWR = 0,17000,5000,5000; // Clockwise circle to reach the point B (X=17 mm,
// Y=5 mm) with a radius of 5 mm
```

This command cannot be executed because the length between the starting and the end point (12 mm) is bigger than the diameter of the circle (10 mm). The UltimET Light automatically introduces a new command line before the above-mentioned command. The executed sequence is:

```
*ILINE = 0,7000,5000,0,0; // linear movement to the point A' (X=7 mm, Y=5 mm)
*ICWR = 0,17000,5000,5000; // clockwise circle to reach point B (X=17 mm, Y=5
// mm) with a radius of 5 mm
```

The circle can be now executed properly.



6.4.1.3 Movement profile commands

A movement profile is applied on the displacement. This profile is defined by a tangential acceleration (ITACC), a tangential deceleration (ITDEC), a tangential speed (ITSPD) and a jerk time (ITJRT).

- **ITSPD command**

The **ITSPD** command allows the user to define the tangential speed to be followed by the next segments. The tangential speed is defined according to the unit format described in §6.1.1.

Syntax:

***ITSPD = ipol_group, tangential speed**

| Parameter | Value | Comment | Units |
|------------------|-----------------------------|-------------------------|----------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| tangential speed | From 0 to $(2^{31}-1)$ (**) | Tangential speed | ufsi (*) |

(*) Refer to §6.1.1 for more information.

(**) It is important to check the maximum speed authorized (K31.#) by the controllers used in the interpolation group. If the tangential speed is too high, some controllers could generate an error (overspeed for example).

The monitorings *M649 and *M669 allow the user to monitor the tangential velocity of the current movement:

| Monitoring | Comment | Units |
|------------|------------------------------------------------|-------|
| *M649 | Tangential velocity, ipol group 0 (unit: ufsi) | ufsi |
| *M669 | Tangential velocity, ipol group 1 (unit: ufsi) | ufsi |

Example:

An example illustrating the ITSPD command is given further in this chapter.

- **ITACC and ITDEC commands**

ITACC and **ITDEC** commands allow the user to define the tangential acceleration and the tangential deceleration to be followed by the next segments. The 'tangential_acceleration' and 'tangential_deceleration' are defined according to the unit format described in §6.1.1.

Syntax:

***ITACC = ipol_group, tangential_acceleration**

***ITDEC = ipol_group, tangential_deceleration**

| Parameter | Value | Comment | Units |
|----------------------------------------------------|-----------------------------|----------------------------------------------------|----------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| Tangential_acceleration or tangential_deceleration | From 0 to $(2^{31}-1)$ (**) | Tangential_acceleration or tangential_deceleration | ufai (*) |

(*) Refer to §6.1.1 for more information.

(**) The maximum value authorized is independent of the motor's capabilities.

The monitorings *M650, *M670 as well as *M651 and *M671 allow the user to monitor the tangential acceleration and deceleration of the current movement:

| Monitoring | Comment | Units |
|------------|---------------------------------------|-------|
| *M650 | Tangential acceleration, ipol group 0 | ufai |
| *M651 | Tangential deceleration, ipol group 0 | ufai |
| *M670 | Tangential acceleration, ipol group 1 | ufai |
| *M671 | Tangential deceleration, ipol group 1 | ufai |

Example:

An example illustrating ITACC and ITDEC command is given further in this chapter.

- **ITJRT command**

The **ITJRT** command allows the user to define the jerk time value for S-curve movements. This jerk time value leads to a jerk limitation and smooths acceleration transitions. On the other hand, compared to the execution time of the same movement with no jerk limitation, there is an extra time equal to this jerk time value

Syntax:

***ITJRT = ipol_group, jerk_time**

| Parameter | Value | Comment | Units |
|------------|------------------|-------------------------|----------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| jerk_time | From 0 to 300 ms | Tangential jerk time | ufti (*) |

(*) Refer to [§6.1.1](#) for more information.

The monitorings *M652 and *M672 allow the user to monitor the tangential jerk time of the current movement:

| Monitoring | Comment | Units |
|--------------|------------------------------------|-------|
| *M652 | Tangential jerk time, ipol group 0 | ufti |
| *M672 | Tangential jerk time, ipol group 1 | ufti |

Example of movement profile:

The graphs below illustrate the different commands relating to the trapezoidal and S-curve movement profiles. Both examples are defined with the same tangential speed, the same tangential acceleration and the same tangential deceleration. The length of the movement is the same in both cases. As the movement profiles are applied along the trajectory, it is not necessary to define if the movement is a line or a circle. For the interpolation group 0, the commands are:

*ITSPD=0, tan_vel

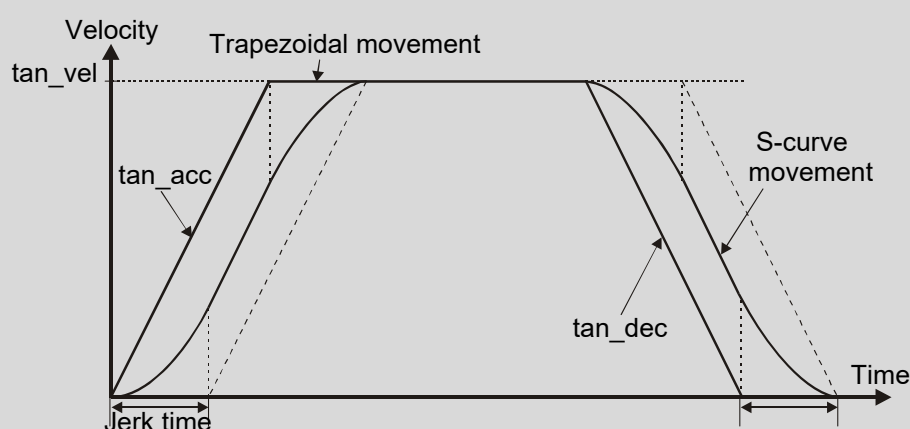
*ITACC=0, tan_acc

*ITDEC=0, tan_dec

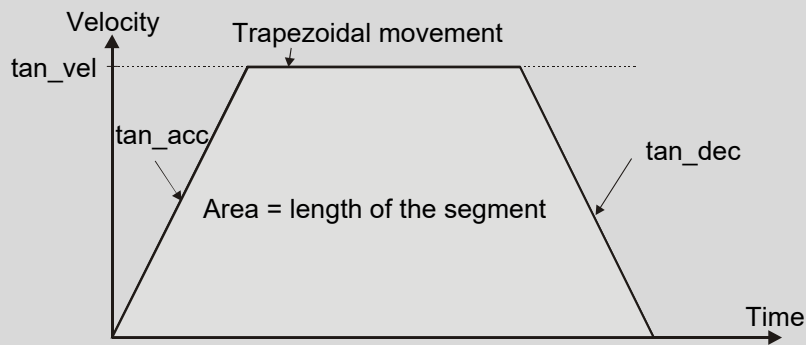
Movement for the first example (trapezoidal movement)

*ITJRT=0, jerk_time

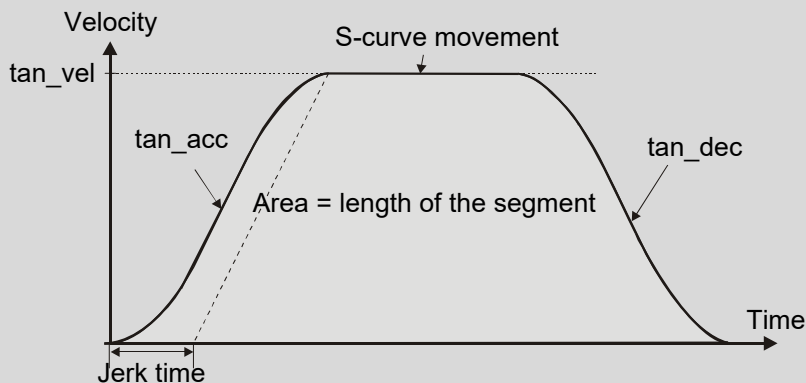
Movement for the second example (S-curve movement)



The first movement profile is trapezoidal. (without jerk time)



The second movement profile is S-curve. (with jerk time)



Both movements do not last the same time. This difference is given by the jerk time. In a S-curve movement, the jerk limitation allows smoother transitions on the mechanical system. The fluidity of the movement is done against the time of execution. The jerk time value is a compromise between the lifetime of the machine and the cycle time.

• ISPD RATE command

The **ISPD RATE** (Interpolation **Speed RATE**) command, which is an alias of the parameter ***K530**, allows the user to reduce the tangential speed. This command acts directly when it is executed. The formula calculating the new value of the tangential speed is:

$$\text{Tangential_speed} \cdot \frac{\text{ISPD RATE}}{100}$$

Syntax:

***ISPD RATE** = <P1> or ***K530** = <P1>

| K | Parameter | <P1> value | Comment | Units |
|-------|-----------|---------------|----------------------------------------------|-------|
| *K530 | <P1> | From 1 to 100 | Reduction of the movement's tangential speed | [%] |

Example:

Here is an example with a XY system with two orthogonal and linear axes. The interpolation group is already initialized on the system. The user unit for position, speed, acceleration and time are respectively the micrometer, micrometer per second, micrometer per squared second and the millisecond.

```

*ITSPD=0,10000;           // tangential speed of 1 cm/s
*ITACC=0,1000000;         // tangential acceleration of 1 m/s2
*ITDEC=0,1000000;         // tangential deceleration of 1 m/s2
*ITJRT=0,10;              // jerk time of 10 ms
*ILINE=0,10000,0,0,0;     // movement from point (0,0) to point (1cm,0)
    
```

Until then, the movement is done with the kinematic and time values (speed = 1 cm/s, acceleration and deceleration = 1 m/s² and jerk time = 10 ms).

```

*ICAM=10; // global diminution to 10% (refer to §6.4.6 for
// more information about ICAM command)
*ILINE=0,10000,10000,0,0; // move to the point (1cm,1cm)
*WTM=0; // waits for the end of the movement

```

For the execution of the second movement, the kinematic and time values are: speed = 0.1 cm/s, acceleration and deceleration = 0.01 m/s² and jerk time = 100 ms.

```

*ISPDRATE=50; // speed diminution of 50%
*ILINE=0,0,0,0,0; // return to the point (0,0)

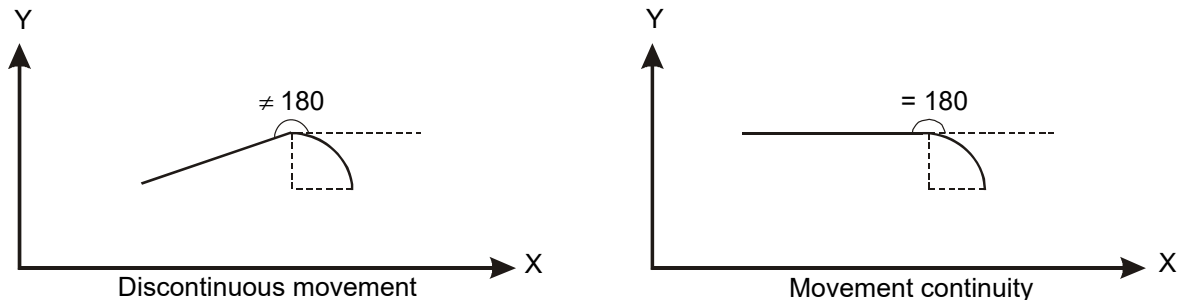
```

For this last movement, the cam is always at 10%. An additional cam is added on the speed only. The speed for this last movement is then equal to 0.05 cm/s.

Remark: The length of the first two movements is identical but the execution time is different.

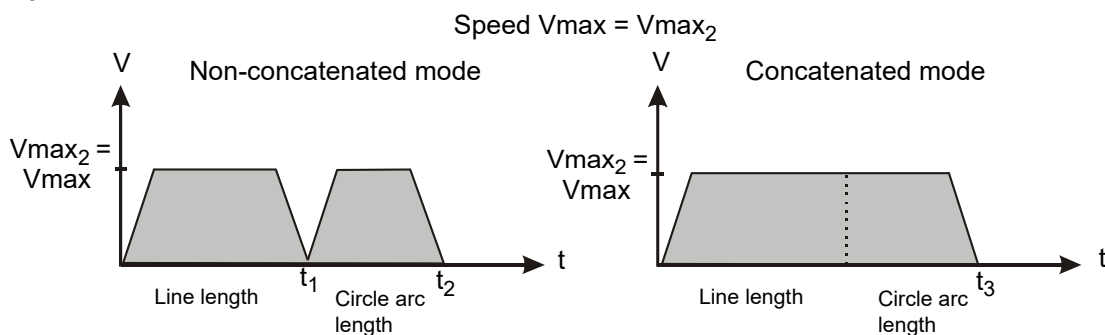
6.4.1.4 Concatenated movement commands

During the generation of the trajectory made up with several segments, the default mode of the UltimET Light is to briefly stop between the segments. This stop allows the axes to physically follow the trajectory. When there is a continuity in the movement (tangent between the end of a segment and the beginning of the next one), the user has the possibility to set the concatenated mode. This mode enables the user to go from a segment to another one without stopping between them. Furthermore, if the speed, acceleration, deceleration and jerk time values are modified, they are automatically taken into account in the new segment.

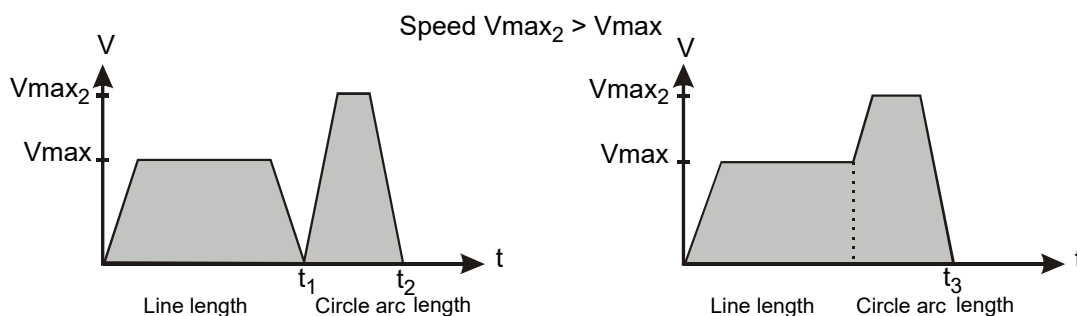


On the above-mentioned graph of the movement continuity, the linear segment can be concatenated with the arc of a circle and then there is no stop between both segments. If the current tangential speed is not modified, the trapezoidal or S-curve profile applies to both segments total length. The deceleration point does not depend on the type of segment currently covered, but on the curvilinear distance still to cover.

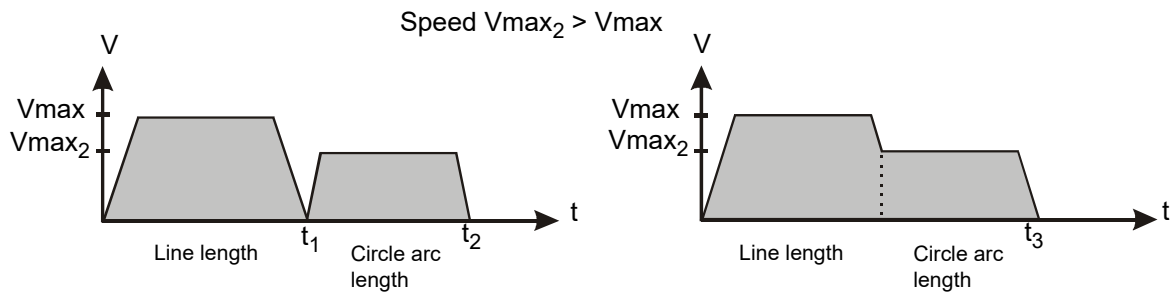
Example:



If a speed command is inserted between both segments, the speed profile acts accordingly. If the new speed is higher than the previous one, the acceleration to reach this speed occurs only at the end of the first segment.



On the other hand, if the new speed is lower, the algorithm provokes a deceleration in such a way that the new speed is reached at the beginning of the new segment.



The **ICONC** command (Interpolation **CONC**atenation) activates the concatenated mode during the trajectory and the **INCONC** command (Interpolation **NO CONC**atenation) stops this mode. A change of the jerk time value between two segments briefly stops the movement, even if the concatenated mode is set.

Remark: The UltimET Light does not detect the continuity of the movement between two segments by itself. If the concatenated mode is active and there is no continuity between two segments, then the UltimET Light will generate the trajectory of the motor with an infinite acceleration. To avoid that, the user has to deactivate the concatenated mode before the second segment.

Syntax:

***ICONC** = ipol_group

***INCONC** = ipol_group

| Parameter | Value | Comment |
|------------|--------|-------------------------|
| ipol_group | 0 or 1 | Interpolated axes group |

The **ICONC** command must be inserted **after** the first movement we want to concatenate and the **INCONC** command is inserted after the last concatenated movement. The trajectory calculator can anticipate up to 20 movements whatever the mode (concatenated or not). It authorizes the execution of short movements in concatenated mode if these movements are already in the interpolation buffer.

Example:

Example with a XY system with two orthogonal and linear axes. Thanks to the **ISSET** command, the node 5 is associated to the X axis and the node 7 to the Y axis. The user unit for position, speed, acceleration and time are micron, micron per second, micron per squared second and microsecond.

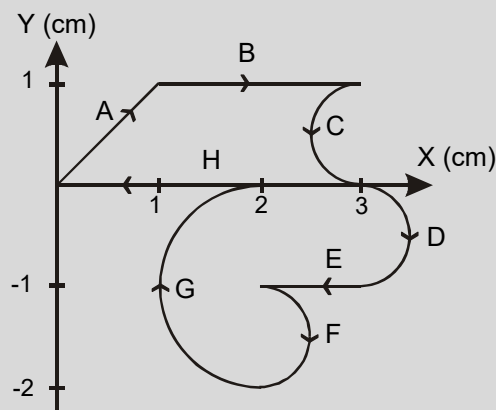
```
*ISET=0,32L,128L,0L,0L; // setting of the interpolated group 0: the masks
                           // containing the nodes numbers are associated to the
                           // X (mask=25) and Y (mask=27) axes.
*K522=1; // user unit for the position is given by *K522 x
          // (10*K523): it is the micron.
*K523=-6;
*K524=-6; // user unit for the speed is the micron per second
          // = 10*K524
*K525=-6; // user unit for the acceleration is the micron per
          // squared second // = 10*K525
*K526 = -6; // user unit for the speed is the microsecond =
          // 10*K526
*IBEGIN = 0; // start of the interpolated mode for the UltimET
             // Light and the corresponding nodes
*ITSPD=0,10000; // tangential speed = 0.01 m/s
*ITDEC=0,1000000; // tangential deceleration = 1 m/s²
*ITACC=0,1000000; // tangential acceleration = 1 m/s²
*ILINE=0,10000,10000,0,0; // segment from point (X=0, Y=0) to point (X=1cm,
                           // Y=1cm)
*ILINE=0,30000,10000,0,0; // segment up to the point (X=3cm, Y=1cm)
*ITSPD=0,12500; // tangential speed = 0.0125 m/s
*ICCW=0,30000,0,30000,5000; // arc of a circle (counter-clockwise) up to the point
                           // (X=3cm, Y=0cm) with the center (Cx=3cm, Cy=0.5cm)
*ICONC=0; // starts the concatenated mode
*ICW=0,30000,-10000,30000,-5000; // arc of a circle (clockwise) up to the point
```

```

// (X=3cm, Y=-1cm) with the center (Cx=3cm, Cy=-
// 0.5cm)
*ITSPD=0,7500; // tangential speed = 0.0075 m/s
*ILINE=0,20000,-10000,0,0; // segment up to the point (X=2cm, Y=-1cm)
*INCONC=0; // stops the concatenated mode
*ITSPD=0,12500; // tangential speed = 0.0125 m/s
*ICW=0,20000,-20000,20000,-15000; // arc of a circle (clockwise) up to the point
// (X=2cm, Y=-2cm) with the center (Cx=2cm, Cy=-
// 1.5cm)
*INCONC=0; // starts the concatenated mode
*ITSPD=0,17500; // tangential speed = 0.0175 m/s
*ICW=0,20000,0,20000,-10000; // arc of a circle (clockwise) up to the point
// (X=2cm, Y=0cm) with the center (Cx=2cm, Cy=-
// 1cm)
*INCONC=0; // stops the concatenated movement
*ITSPD=0,10000; // tangential speed = 0.01 m/s
*ILINE=0,0,0,0,0,0; // segment up to the point (X=0cm, Y=0cm)

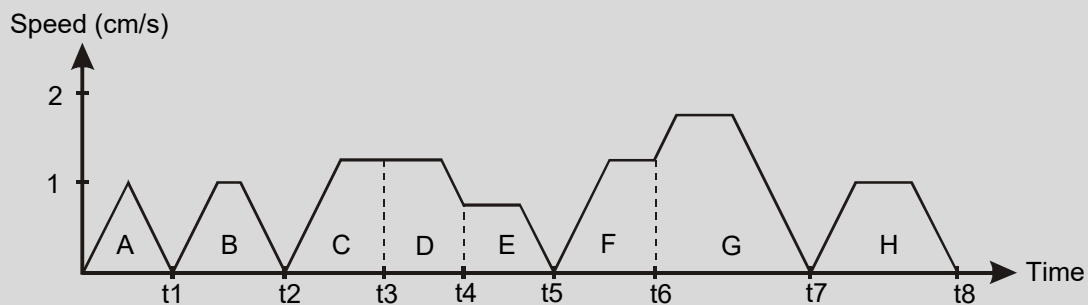
```

The generated trajectory is:



In the above-mentioned figure, the first three movements are not concatenable. At the end of the first two movements, a stop has to be done to prevent the motor from having an 'infinite' theoretical acceleration with the intention to catch up with the other motor which did not stop. The movement continuity between the segments C, D and E as well as F and G authorizes the concatenated mode.

The figure below shows the speed profile applied along the trajectory with a trapezoidal movement:



Remark:

When a speed command introduces a reduced speed in a chain of concatenated movements (between segments D and E), the calculator anticipates this speed change. The speed already decreases in the previous segment so that the speed at the beginning of the next segment will be good. On the other hand, if the speed increases, the calculator waits for the end of the current segment (current speed is lower) to accelerate up to the next defined speed (segment F and G). This technique is used to respect the maximum speed along the entire segment (between segments D and E for example). On the other hand, the real speed can be lower than this maximum fixed speed if the chain of the segments requires it (anticipation of the maximal speed of the next segment).

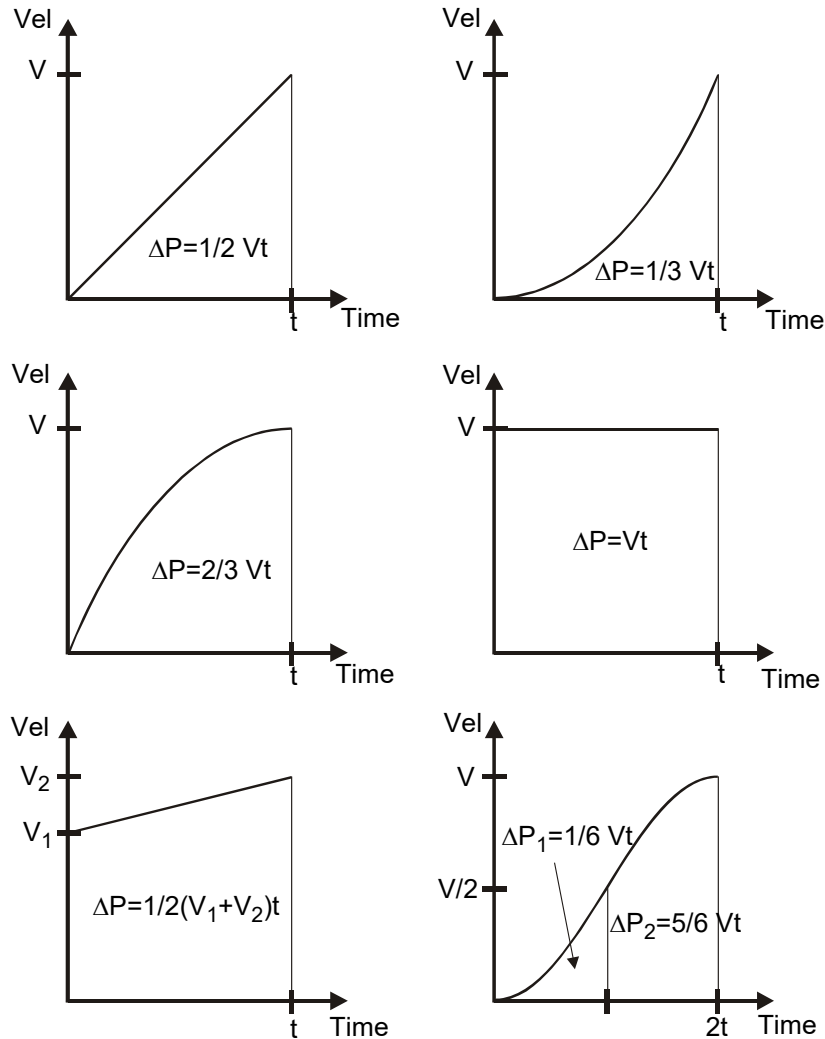
6.4.2 PVT (Position-Velocity-Time) mode

The type of interpolation described in this chapter offers more flexibility (than the G-code based movement) while leaving the user the definition of the movement profile and the generation of the trajectory. The **PVT**

movement describes a segment by giving its arrival point B (the starting point A is the current point), its speed vector at this point and the time to run this segment. The speed profile is then defined by the user who can execute the trajectory according to his needs. On the other hand, this mode requires more complex calculations from the user.

The algorithm calculating the trajectory between two points is based on a third-degree polynomial. A speed continuity is ensured thanks to the degree of the polynomial; on the other hand, the user has to manage the continuity of the acceleration and to limit the jerk when he wants to make a trajectory.

Profile examples:



Trajectory example



- **IPVT command**

The **IPVT** command (Interpolation **P**osition-**V**elocity-**T**ime) allows the user to execute an interpolated movement on 4 axes according to a trajectory defined by a third-degree polynomial. There are some limitations. The time parameter of this command must be a multiple of 1sti (100 μ s). The minimum time per PVT segment is 1sti and any value under this time would lead to an error or an uncontrolled trajectory. The maximum time per PVT segment is 2.3 second and using higher time segment would lead to uncontrolled trajectory.

Syntax:

***IPVT = ipol_group, coord_x, coord_y, coord_z, coord_θ, vel_x, vel_y, vel_z, vel_θ, time**

| Parameter | Value | Comment | Units |
|------------|------------------------------------------|-----------------------------------|----------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| coord_x | From -distance_max to +distance_max (**) | x coordinates of the target point | ufpi (*) |
| coord_y | From -distance_max to +distance_max (**) | y coordinates of the target point | ufpi (*) |
| coord_z | From -distance_max to +distance_max (**) | z coordinates of the target point | ufpi (*) |
| coord_θ | From -distance_max to +distance_max (**) | θ coordinates of the target point | ufpi (*) |
| vel_x | From -vel_max to vel_max (***) | Speed on x of the target point | ufsi (*) |
| vel_y | From -vel_max to vel_max (***) | Speed on y of the target point | ufsi (*) |
| vel_z | From -vel_max to vel_max (***) | Speed on z of the target point | ufsi (*) |
| vel_θ | From -vel_max to vel_max (***) | Speed on θ of the target point | ufsi (*) |
| time | From 0 to time_max (****) | Time of the movement | ufti (*) |

(*) and (**): Refer to [§6.1.1](#) for more information.

(***): vel_max is equal to $(2^{31}-1)$

(****): max. time is 2.3 sec

The PVT interpolation algorithm keeps expecting PVT points continuously until it receives a last point with all coordinate having speed = 0 (*IPVT = ipol_group, pos_x, pos_y, pos_z, pos_t, 0, 0, 0, 0, time_move;).

In the case where the PVT buffer is found empty and the last point contained a position with a speed !=0, the UltimET generates an emergency safe stop of the axes with a pre-defined deceleration *K546. The braking time is the maximum time of all axes (time_brake = speed_axis_i/*K546:i). This ensures that all axes are stopped at the same time!

The final position reached is defined with respect to the last valid point received with a position delta of delta_P = $1/3 V*t$, where V is the speed at the last valid point and "t" the braking time. Such is the profile for the emergency braking.

It should be also added that the UltimET deactivates the interpolation mode at the end of the emergency braking. Therefore, no *IEND command must be sent afterwards. In effect that would lead to an error.

If such list of command are sent:

```
*IPVT = group_numb, 70,-75,0,0,0,-50,0, 0, 500;
```

```
*wtm = group_numb;
```

```
*IEND = group_numb;
```

It leads to an UltimET error 1032 (local command not acknowledged).

When an emergency brake starts, a warning is raised (warning code 18) and a bit #2 is set in *M519:ipol_group (Interpolation Status register)). The warning and the interpolation status register bit are cleared either after a *RST command is called or at the next call of *IBEGIN=ipol_group.

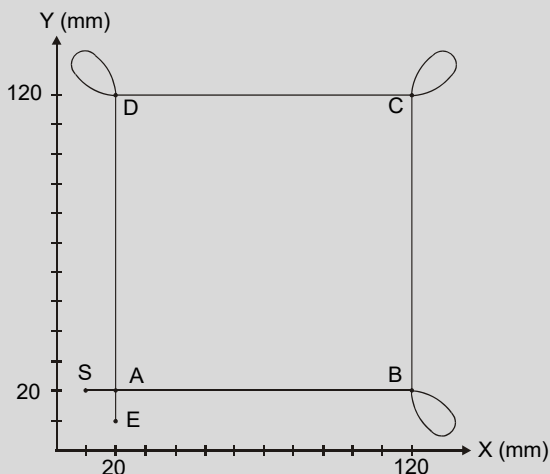
Refer to §11. for the list of warnings and to §4.14.3 for the description of *M519.

| Parameter | Comment | Units |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| *K546 | Deceleration value for each axis in case of emergency braking in PVT mode (depths 0 = axis X of group 0, 1 = Y of group 0, ..., 4 = X of group 1, ...) | ufai |

Example:

In a XY system, a square form is cut into a metal piece. The feed rate is fixed to 50 mm/s and must be followed. The four corner points of the square form are A (X=20mm, Y=20mm), B (X=120mm, Y=20mm), C (X=120mm, Y=120mm) and D (X=20mm, Y=120mm). To respect this feed rate, the trajectory is defined as follow: the coordinates of the starting point S are: X=10mm, Y=20mm, the coordinates of the end point E are: X=20mm, Y=10mm, the user position unit is the millimeter and the user speed unit is in mm/s.

```
*IPVT=0,20,20,0,0,50,0,0,0,200; // segment time is 200 ms
*IPVT=0,120,20,0,0,50,0,0,0,2000; // segment time is 2 s
*IPVT=0,120,20,0,0,0,50,0,0,1000; // segment time is 1 s
*IPVT=0,120,120,0,0,0,50,0,0,2000; // segment time is 2 s
*IPVT=0,120,120,0,0,-50,0,0,0,1000; // segment time is 1 s
*IPVT=0,20,120,0,0,-50,0,0,0,2000; // segment time is 2 s
*IPVT=0,20,120,0,0,0,-50,0,0,1000; // segment time is 1 s
*IPVT=0,20,20,0,0,0,-50,0,0,2000; // segment time is 2 s
*IPVT=0,20,10,0,0,0,0,0,0,200; // segment time is 200 ms
```



• ISPD RATE command

The **ISPD RATE** (Interpolation **Speed** **R**ATE) command, which is an alias of the parameter *K530, allows the user to reduce the tangential speed. This command acts directly when it is executed. The formula calculating the new value of the tangential speed is:

$$\text{Tangential_speed} \cdot \frac{\text{ISPD RATE}}{100}$$

Syntax:

*ISPD RATE = <P1>

| Parameter | <P1> value | Comment | Units |
|-----------|---------------|----------------------------------------------|-------|
| <P1> | From 1 to 100 | Reduction of the movement's tangential speed | % |

The values corresponding to the IBRK, ICONT and ISPD RATE commands are calculated according to the time value of the parameter *K561.

| Parameter | Comment | Units |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| *K561 | Time value used for a speed rate change from 100% to 0% along the trajectory in PVT and PT mode. The depths correspond to the ipol group numbers | ufti |

Example:

During a movement made of PVT segments, a brake command (IBRK) will stop the movement along the trajectory within the time defined in parameter *K561. To restart the movement at full speed (ICONT command), the same time will be needed.

6.4.2.1 On-fly trajectory change

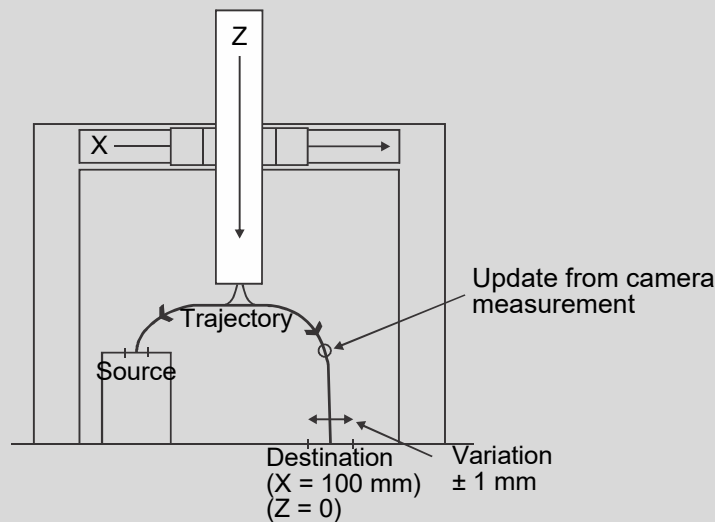
In many applications, the final position of a tool is known when the movement has already started. For example, in pick and place applications, a camera determines the exact position of a component just before the gripping tool puts it. The trajectory of the gripping tool has to be modified 'on-fly' to properly put the component.

The PVT mode is able to modify the trajectory on-fly. In the above-mentioned example of the IPVT command, the parameters are immediate values. To give to the user a complete flexibility, the IPVT command can also contain variables.

The PVT mode uses a special set of registers: the Y registers ($0 \leq Y \leq 255$ on 4 depths). Thus, in the IPVT command, each command parameter can be either an immediate value or a Y register (except the interpolation group parameter). An immediate value cannot be modified but an Y register can be modified at any time. In the IPVT command, Y registers can only be used at depth 0.

Example:

In a XZ system which realizes pick and place operations, a gripping tool is moved by two axes. The gripping tool takes a piece from a source position to a destination position. The coordinate of the destination point is not so precise on the X axis (± 1 mm).

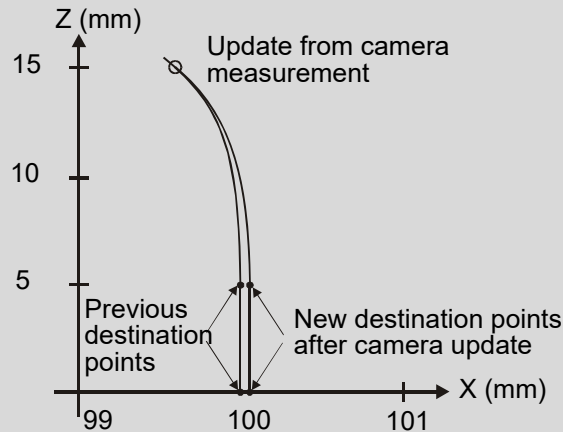


During the execution of the trajectory from the source point to the destination point, a camera helps to determine the exact position of the piece at the destination point. Here is an example of the last commands sent to the UltimET Light to reach the destination point. This destination point has to be reached vertically to properly put the piece. The user position unit is in micron, user speed unit is in millimeter/second and the user time unit is in millisecond.

```
*IPVT=0,Y10,0,5000,0,0,10,0,0,15; // with *Y10=100000 corresponding to X=100mm
*IPVT=0,Y10,0,0,0,0,0,0,0,5;
```

When the information from the camera arrive (the new X coordinate is 100.1 mm), the first command of the two mentioned above is running. The X position is updated thanks to the *Y10 variable (*Y10=100100 corresponding to X=100.1mm).

The target has been modified and is changed on-fly.



In this example, only one variable has changed. If the Z coordinate of the final point is also a variable, it is not possible to change both variables at the same time. In some special cases, this restriction can be a limitation and a solution to this limitation is given [§6.4.2.2](#).

6.4.2.2 Simultaneous update of several Y registers

The **IPVTUPDATE** command allows to simultaneously update a block (or a part of it) of 32 consecutive Y registers. The number of the first Y register of the block is stored in the parameter ***K560**.

The depth 0 of the Y registers contains the active values describing the trajectory. Other depths may be used to temporarily store new values. The IPVTUPDATE command copies those values from a specific depth to the depth 0 (for a block of 32 Y registers). When the copy is done, the values of depth 0 have changed and the trajectory is modified.

The mask parameter defines which registers, in the block of 32 Y registers, would be updated by the IPVTUPDATE command. This parameter is a 32 bits word and each bit defines one register.

Syntax:

***IPVTUPDATE = depth, mask**

| Parameter | Value | Comment |
|-----------|--------------------------|--------------------------------------|
| Depth | From 0 to 3 | Depth where the new value are stored |
| Mask | From 0 to ($2^{32}-1$) | Mask of the modified Y registers |

Example:

The previous example ([§6.4.2.1](#)) only modifies the X position. In this example, Z position can also be modified and then the commands are:

```
*K560=10;           // Beginning of the block of Y registers used in the next
                    // IPVTUPDATE command
*IPVT=0,Y10,0,Y11,0,0,0,0,0,0,15;
*IPVT=0,Y10,0,0,0,0,0,0,0,0,15;
```

with

| Register \ depth | 0 | 1 | 2 | 3 |
|------------------|--------|---|---|---|
| Y10 | 100000 | 0 | 0 | 0 |
| Y11 | 5000 | 0 | 0 | 0 |

The camera updates the positions for X and Z:

```
*Y10:1=100100;
*Y11:1=5200;
```

The value of the registers are:

| Register \ depth | 0 | 1 | 2 | 3 |
|------------------|--------|--------|---|---|
| Y10 | 100000 | 100100 | 0 | 0 |
| Y11 | 5000 | 5200 | 0 | 0 |

The target positions have not changed yet. The first command target is (X=100mm, Z=5mm), the second one is (X=100mm, Z=0mm).

Now the following command is sent:

```
*IPVTUPDATE=1,0x03; // update of the Y registers pointed by *K560 (Y10) and
                        // included in the mask 0x03 = (011)binary i.e. the two
                        // first (Y10 and Y11).
```

The values of the registers are now:

| Register \ depth | 0 | 1 | 2 | 3 |
|------------------|--------|--------|---|---|
| Y10 | 100100 | 100100 | 0 | 0 |
| Y11 | 5200 | 5200 | 0 | 0 |

The copy has been done and the target positions have been updated. The first command target now is (X=100.1mm, Z=5.2mm), the second one is (X=100.1mm, Z=0mm).

Caution: A parameter change in the PVT mode can induce great modifications of the acceleration in the trajectory generation.

6.4.3 PT (Position-Time) mode

As well as the PVT mode, the type of interpolation described in this chapter offers more flexibility (than the G-code based movement) while leaving the user the definition of the movement profile and the generation of the trajectory. The **PT** movement describes a segment by giving its arrival point B (the starting point A is the current point) and the time to run this segment. The speed profile is then automatically defined by the following principle:

- If only one segment is processed, the initial and final velocity is 0.
- If more than one segment is processed (standard case), the following velocity profile rule is applied:
 - The first segment initial velocity is zero
 - The first segment final velocity is: $vel_pt(0) = (pos_pt(1) - pos_pt(0)) / time_pt(1)$
 With
 $vel_pt(0)$ is the first segment final velocity,
 $pos_pt(0)$ is the target point of the current segment,
 $pos_pt(1)$ is the target point of the next segment,
 $time_pt(1)$ is the next segment time
 - The next segment final velocity is: $vel_pt(0) = (pos_pt(1) - pos_pt(-1)) / (2 * time_pt(1))$
 With
 $vel_pt(0)$ is the first segment final velocity,
 $pos_pt(-1)$ is the initial point of the current segment,
 $pos_pt(1)$ is the target point of the next segment,
 $time_pt(1)$ is the next segment time
 - The last segment final velocity is 0

- **IPT command**

The **IPT** command (Interpolation Position-Time) allows the user to execute an interpolated movement on 4 axes. As well as for PVT mode, there are some limitations. The time parameter of this command must be a

multiple of 1st (100 μ s). The minimum time per PT segment is 1st and any value under this time would lead to an error or an uncontrolled trajectory. The maximum time per PVT segment is 2.3 second and using higher time segment would lead to uncontrolled trajectory.

Syntax:

***IPT = ipol_group, coord_x, coord_y, coord_z, coord_θ, time**

| Parameter | Value | Comment | Units |
|------------|------------------------------------------|-----------------------------------|----------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| coord_x | From -distance_max to +distance_max (**) | x coordinates of the target point | ufpi (*) |
| coord_y | From -distance_max to +distance_max (**) | y coordinates of the target point | ufpi (*) |
| coord_z | From -distance_max to +distance_max (**) | z coordinates of the target point | ufpi (*) |
| coord_θ | From -distance_max to +distance_max (**) | θ coordinates of the target point | ufpi (*) |
| time | From 0 to time_max (***) | Time of the movement | ufti (*) |

(*) and (**): Refer to [§6.1.1](#) for more information.

(***): max. time is 2.3 sec.

The PVT interpolation algorithm keeps expecting PVT points continuously until it receives a last point with all coordinate having speed = 0 (*IPVT = ipol_group, pos_x, pos_y, pos_z, pos_t, 0, 0, 0, 0, time_move;).

In the case where the PVT buffer is found empty and the last point contained a position with a speed !=0, the UltimET generates an emergency safe stop of the axes with a pre-defined deceleration *K546. The braking time is the maximum time of all axes (time_brake = speed_axis_i/*K546:i). This ensures that all axes are stopped at the same time!

The final position reached is defined with respect to the last valid point received with a position delta of $\Delta P = 1/3 V \cdot t$, where V is the speed at the last valid point and "t" the braking time. Such is the profile for the emergency braking.

It should be also added that the UltimET deactivates the interpolation mode at the end of the emergency braking. Therefore, no *IEND command must be sent afterwards. In effect that would lead to an error.

If such list of command are sent:

```
*IPVT = group_numb, 70, -75, 0, 0, 0, -50, 0, 0, 500;
```

```
*wtm = group_numb;
```

```
*IEND = group_numb;
```

It leads to an UltimET error 1032 (local command not acknowledged).

When an emergency brake starts, a warning is raised (warning code 18) and a bit #2 is set in *M519:ipol_group (Interpolation Status register)). The warning and the interpolation status register bit are cleared either after a *RST command is called or at the next call of *IBEGIN=ipol_group.

Refer to [§11](#). for the list of warnings and to [§4.14.3](#) for the description of *M519.

| Parameter | Comment | Units |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| *K546 | Deceleration value for each axis in case of emergency braking in PVT mode (depths 0 = axis X of group 0, 1 = Y of group 0, ..., 4 = X of group1, ...) | ufai |

• ISPD RATE command

The **ISPD RATE** (Interpolation **Speed** **R**ATE) command, which is an alias of the parameter ***K530**, allows the user to reduce the tangential speed. This command acts directly when it is executed. The formula calculating

the new value of the tangential speed is:

$$\text{Tangential_speed} \cdot \frac{\text{ISPDRATE}}{100}$$

Syntax:

***ISPDRATE = <P1>**

| Parameter | <P1> value | Comment | Units |
|-----------|---------------|----------------------------------------------|-------|
| <P1> | From 1 to 100 | Reduction of the movement's tangential speed | [%] |

The values corresponding to the IBRK, ICONT and ISPDRATE commands are calculated according to the time value of the parameter ***K561**.

| Parameter | Comment | Units |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| *K561 | Time value used for a speed rate change from 100% to 0% along the trajectory in PVT and PT mode. The depths correspond to the ipol group numbers | ufti |

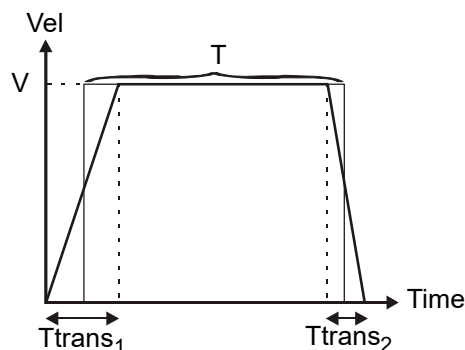
6.4.4 ULM mode

6.4.4.1 Introduction

The **ULM** (**U**niversal **L**inear **M**ovement) allows the user to define a trajectory made up of a series of linear segments put end to end. By default, the segments are automatically concatenated which means that there is no stop between the segments. The transition between 2 successive segments is defined by a parameter of time. During this time, a segment ends while the other begins which means there is a smooth transition between two segments. Therefore, the real trajectory does not necessarily go through the defined points except for the starting and end point which are respected.

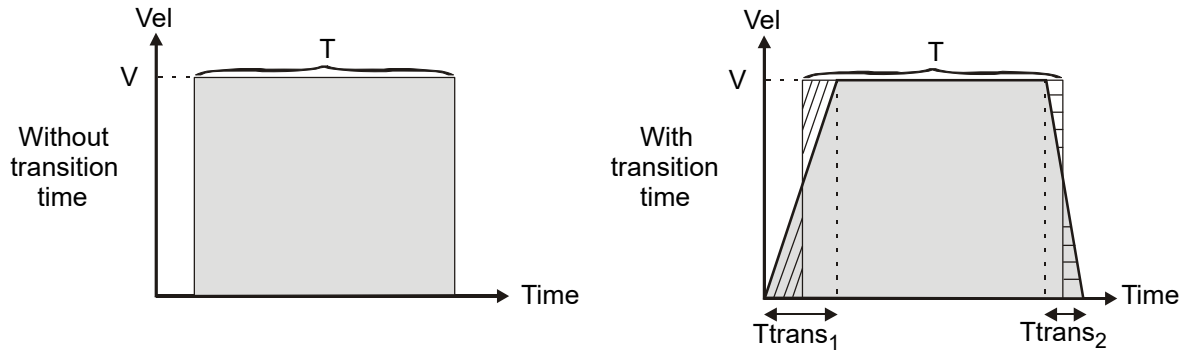
A segment is thus defined by 5 different parameters:

- the starting point
- the first transition time: T_{trans_1}
- the speed of the segment: V
- the second transition time: T_{trans_2}
- the end point

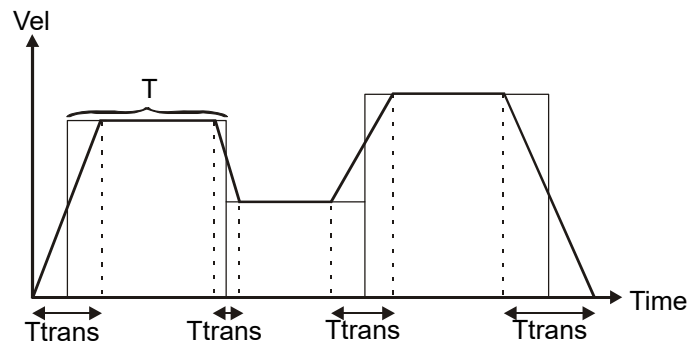


$$T = \text{Distance} / \text{speed}$$

Without transition time (T_{trans}) and at constant speed (V), the speed profile would be a rectangle. The stroke, corresponding to the speed integral, is represented by the area under the speed profile. As long as this area is kept, the final position is ensured. When the transition time is added, the area does not change.



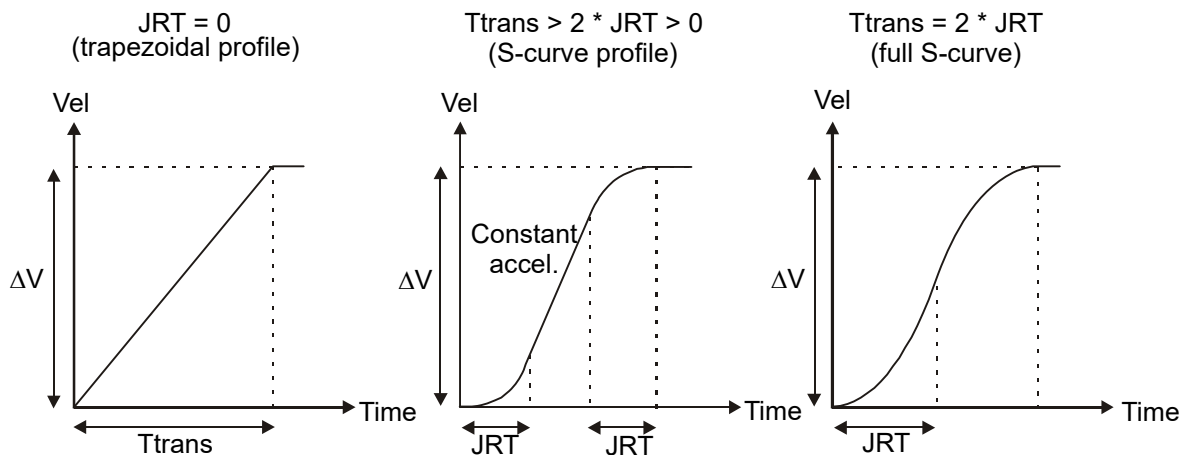
If there are many concatenated segments, the second transition time of a segment is identical and coincide with the first transition time of the following segment. On a three segments trajectory, the speed profile could be as follows:



A jerk time can be applied on any transition and can be different on each of them. Nevertheless, this jerk time is not applied like the one in ETEL's controllers. In this mode, the jerk time is included in the transition time and not added to the movement time.

The acceleration on each axis depends on:

- the speed variation: ΔV
- the transition time: T_{trans}
- the jerk time: JRT

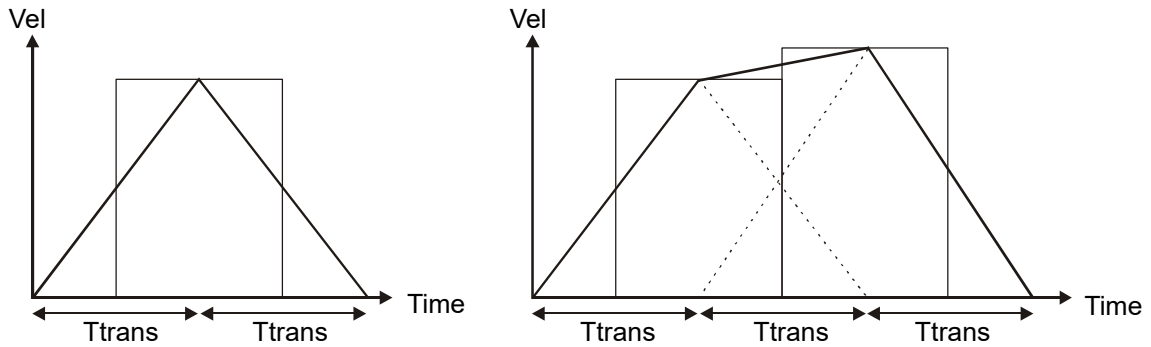


Remark: If the value of T_{trans} is lower than $2 * JRT$, then it is automatically forced to be: $T_{trans} = 2 * JRT$

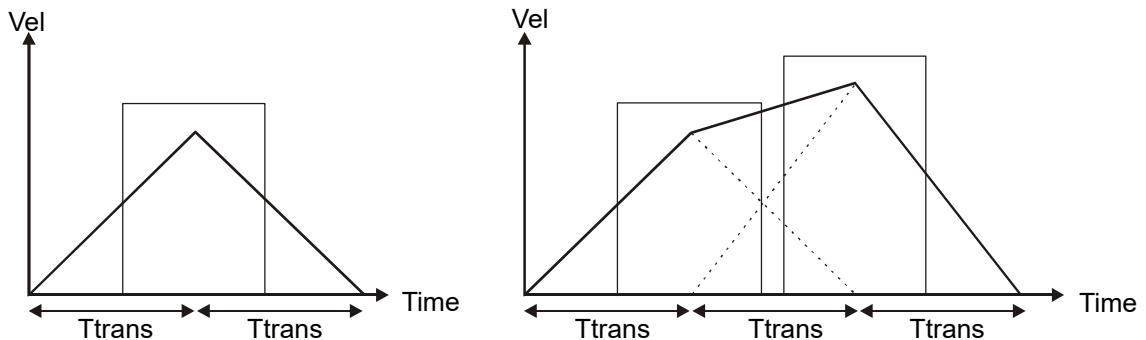
The maximum acceleration (Acc_{max}) during this transition is thus equal to: $Acc_{max} = \Delta V / (T_{trans} - JRT)$.

Profile examples

- If the two transition times (T_{trans}) are equal to the movement time, the speed profile for one and two linear segments respectively is as follows:

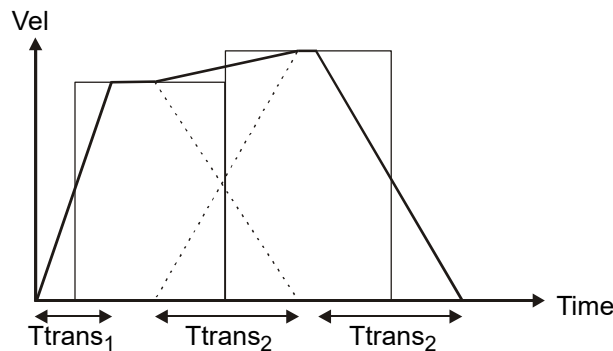


- If the transition times (T_{trans}) are too important (acceleration too small), the speed profile for one and two linear segments respectively is as follows:



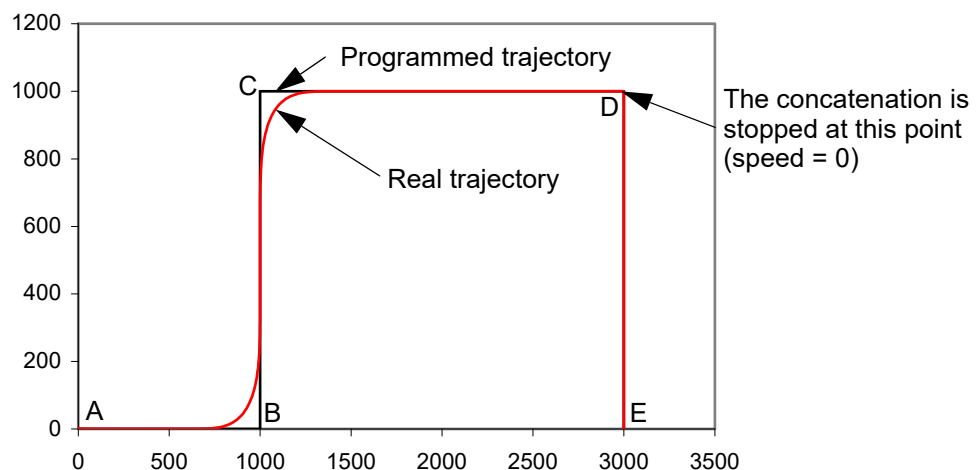
Here is a segment needing T time to be executed at constant speed. If $T < (T_{trans1} + T_{trans2}) / 2$ then T is forced to be: $T = (T_{trans1} + T_{trans2}) / 2$ and the speed has to be recalculated.

- If the transition times of the first and second linear segments are different, the speed profile is as follows:

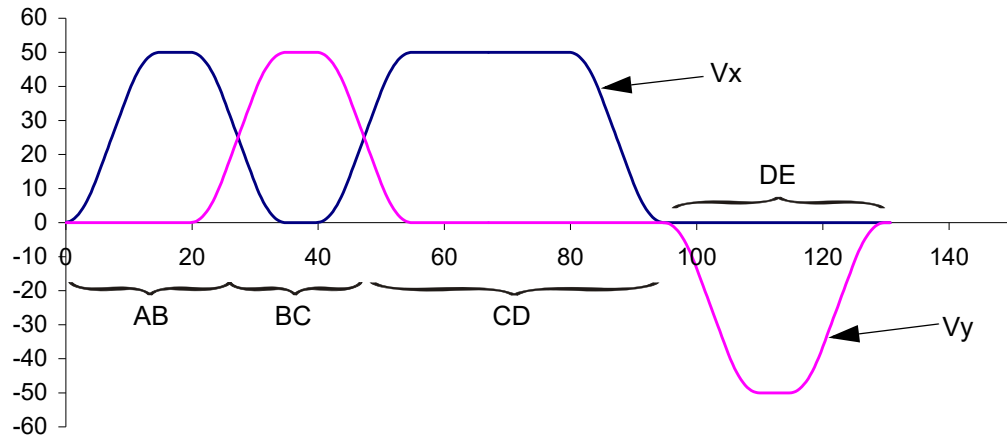


Trajectory example

Here is an example showing that the real trajectory does not necessarily go through the defined points except for the starting and end point which are respected as well as the ones forced to be respected.



Here is the speed profile, V_x and V_y , corresponding to the above-mentioned trajectory.



6.4.4.2 ULM commands

- **IULINE command**

The **IULINE** command defines the destination point of the segment and, if desired, the second transition time of this segment (T_{trans2} and JRT_2).

Syntax:

***IULINE = ipol_group, coord_x, coord_y, coord_z, coord_θ, (T_{trans2} , JRT_2)**

| Parameter | Value | Comment | Units |
|--------------|------------------------------------------|-----------------------------------------|----------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| coord_x | From -distance_max to +distance_max (**) | x coordinate of the target point | ufpi (*) |
| coord_y | From -distance_max to +distance_max (**) | y coordinate of the target point | ufpi (*) |
| coord_z | From -distance_max to +distance_max (**) | z coordinate of the target point | ufpi (*) |
| coord_θ | From -distance_max to +distance_max (**) | θ coordinate of the target point | ufpi (*) |
| T_{trans2} | From 0 to time_max (***) | Second transition time | ufti (*) |
| JRT_2 | From 0 to time_max (***) | Jerk time of the second transition time | ufti (*) |

(*) and (**): Refer to §6.1.1 for more information.

(***): Defined in the IWTT command (§6.4.6).

The T_{trans2} and JRT_2 parameters can be omitted (command with only 5 parameters instead of 7) if the new transition and jerk times are identical to the previous ones. If they are mentioned and if $T_{trans2} < 2 * JRT_2$, T_{trans2} is forced to be equal to $2 * JRT_2$.

Before sending the first IULINE command to the UltimET Light, it is necessary to define the initial tangential speed with the IUSPEED command, as well as the initial transition and jerk times with the IUTIME command.

- **IUSPEED command**

The **IUSPEED** command defines the tangential speed on the trajectory. The axes involved in this trajectory are specified by the IUSPDMASK command.

Syntax:

***IUSPEED = ipol_group, tangential_speed**

| Parameter | Value | Comment | Units |
|------------------|----------------------|------------------------------------|----------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| tangential_speed | From 0 to $2^{31}-1$ | Tangential speed on the trajectory | ufsi (*) |

(*): Refer to §6.1.1 for more information.

The speed on each interpolated axis, belonging to the *IUSPDMASK command, corresponds to the projection of the tangential speed.

To limit the speed for each interpolated axis, the parameter *K545 contains the maximal speed value of these axes. If the theoretical speed of an interpolated axis (X, Y, Z or θ) exceeds its corresponding *K545 value, the speed of all the interpolated axes is proportionally reduced.

| Parameter | <P1> value | Default value | Comment | Units |
|--------------------|----------------------|---------------|--------------------------------------------------------------------------------------------------------------------------------|-------|
| *K545:depth = <P1> | From 0 to $2^{31}-1$ | 0 | Maximal speed per interpolated axis in ULM mode. Depth 0 = X axis, depth 1 = Y axis, depth 2 = Z axis, depth 3 = θ axis | ufsi |

Remark: The default value (0) corresponds to no limitation.

• IUSPDMASK command

The IUSPDMASK command defines the axes of the trajectory where the tangential speed is applied (defined by the *IUSPEED)

Syntax:

***IUSPDMASK = ipol_group, axes_mask_for_tan_group**

| Parameter | Value | Comment | Units |
|-------------------------|---------------|-------------------------------------------------------|-------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| axes_mask_for_tan_group | From 0 to 0xF | Mask of the interpolated axes defining the trajectory | - |

| Axis | axes_mask_for_tan_group value |
|----------|-------------------------------|
| X | Bit 0 => 0x1 |
| Y | Bit 1 => 0x2 |
| Z | Bit 2 => 0x4 |
| θ | Bit 3 => 0x8 |

By default, the value is 0x7 which means that the trajectory is made up of the X, Y and Z axes (the θ axis follows the other axes).

Example:

The current point is O (X=0, Y=0, Z=0, θ =0) and the destination point is P (2, 3, 4, 0). The tangential speed applied to the X, Y and Z axes is equal to 10. The projected speeds are calculated as follows:

Length of the segment: $\text{length} = \sqrt{2^2 + 3^2 + 4^2} = \sqrt{29} \approx 5.385$

Time for the segment: $\text{time} = \text{length} / \text{speed} = 0.5385$

Speed on the X axis: $\text{speed_X} = \text{length_on_X} / \text{time} = 2 / 0.5385 = 3.714$

Speed on the Y axis: $\text{speed_Y} = \text{length_on_Y} / \text{time} = 3 / 0.5385 = 5.571$

Speed on the Z axis: $\text{speed_Z} = \text{length_on_Z} / \text{time} = 4 / 0.5385 = 7.248$

If an axis does not belong to the IUSPDMASK command, the speed on this axis is calculated in order to follow the other axes. If in the previous example, the destination point P is (2, 3, 4, 8) instead of P (2, 3, 4, 0), the speed on the θ axis is:

Length of the segment: $\text{length} = \sqrt{2^2 + 3^2 + 4^2} = \sqrt{29} \approx 5.385$

Time for the segment: $\text{time} = \text{length} / \text{speed} = 0.5385$

Speed on the θ axis: $\text{speed } \theta = \text{length_on_}\theta / \text{time} = 8 / 0.5385 = 14.856$

The movement time is defined by X, Y and Z axes. As the θ axis stroke must be covered during this movement time, the speed of this axis is calculated to respect it. The speed for this axis is thus bigger than the tangential speed.

• IUTIME command

The **IUTIME** command is used before the first IULINE command to define the first transition and jerk times. This command defines also the transition and jerk times for the next transitions as long as parameters number 6 and 7 of the IULINE command are omitted or no other IUTIME command is sent. If this command is sent between two segments, its parameters can be applied in two different ways:

- if the segments are concatenated (it is the case by default), the parameters are applied to the second transition of the next segment.
- if the segments are not concatenated (refer to the IUNOCONC command for more information), the parameters are already applied to the first transition of the next segment.

Syntax:

***IUTIME = ipol_group, Ttrans, JRT**

| Parameter | Value | Comment | Units |
|------------|-------------------------|-------------------------|----------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| Ttrans | From 0 to time_max (**) | Transition time | ufti (*) |
| JRT | From 0 to time_max (**) | Jerk time | ufti (*) |

(*): Refer to [§6.1.1](#) for more information.

(**): Defined in the IWTT command [§6.4.6](#).

Remark: If $Ttrans < 2 * JRT$, Ttrans is forced to be equal to $2 * JRT$.

• IURELATIVE command

The **IURELATIVE** command allows the user to enable or disable the relative mode for the universal linear movement that is to say to give the destination point in absolute or relative coordinates.

Syntax:

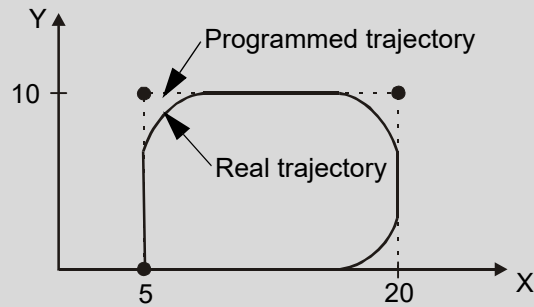
***IURELATIVE = ipol_group, relative_mode**

| Parameter | Value | Default value | Comment |
|---------------|--------|---------------|---------------------------|
| ipol_group | 0 or 1 | - | Interpolated axes group |
| relative_mode | 0 or 1 | 0 | Absolute or relative mode |

The relative mode is enabled if the 'relative_mode' parameter is set (at 1) and disabled if reset (at 0). By default, the absolute mode is set ('relative_mode' parameter = 0).

Example:

```
*IUTIME=0,500,200;
*IUSPEED=0,10;
*IULINE=0,20,0,0,0;
*IULINE=0,20,10,0,0;
*IURELATIVE=0,1;
*IULINE=0,-15,0,0,0;
*IULINE=0,0,-10,0,0;
```



• IUNOCONC command

By default, the segments are concatenated in the ULM mode. The **IUNOCONC** command disables the concatenation between two segments (the one preceding the command and the one following the command). This command is only valid for one transition and has to be put between all the segments if the user wants the movement to stop between each segment.

Syntax:

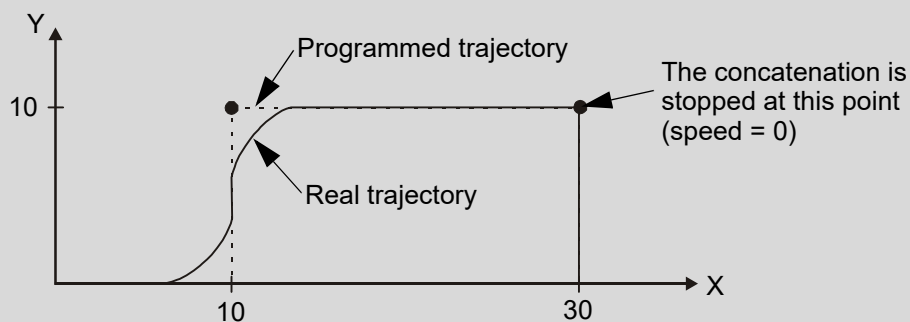
***IUNOCONC = ipol_group**

| Parameter | Value | Comment |
|------------|--------|-------------------------|
| ipol_group | 0 or 1 | Interpolated axes group |

Remark: If the IUNOCONC command is used between two segments and without modification of any transition time, the first transition time of the second segment is equal to the second transition time of the first segment.

Example:

```
*IUTIME=0,500,200;
*IUSPEED=0,10;
*IULINE=0,10,0,0,0;
*IULINE=0,10,10,0,0;
*IULINE=0,30,10,0,0;
*IUNOCONC=0;
*IULINE=0,30,0,0,0;
```



6.4.5 Braking commands

6.4.5.1 IBRK and ISTP commands

In an emergency situation, it may be necessary to immediately stop the current movement. Two commands are available for the interpolation: IBRK and ISTP commands.

The **IBRK** command (Interpolation **BRaKe**) stops the current movement (the braking is done along the trajectory) with a defined deceleration:

- In G-code based mode, this tangential deceleration is defined by the last one programmed

- In ULM mode, this deceleration is defined by the value of parameters ***K532** and ***K533**
- In PVT and PT mode, this deceleration time is defined by the value of parameter ***K561**

| Parameter | <P1> value | Default value | Comment | Units |
|-----------|-------------------------|---------------|-------------------------------------------------------------------------------|----------|
| *K532 | From 0 to time_max (**) | 0 | Deceleration time used when the IBRK command is sent during an IULINE command | ufti (*) |
| *K533 | From 0 to time_max (**) | 0 | Jerk time used when the IBRK command is sent during an IULINE command | ufti (*) |

(*): Refer to [§6.1.1](#) for more information.

(**): Defined in the IWTT command [§6.4.6](#).

Remark: After a IBRK command, waiting for the complete stop of the axes may be done using the ***WTM=ipol_group** command. Do not close the interpolation group (***IEND**) before the end of the movement otherwise it would be similar to a HLT command.

The **ISTP** command (Interpolation **STOp**), valid for all interpolation modes, stops the movement with an infinite deceleration. This command stops the movement instantly and may provoke an error on the moving axes. This command is only used in case of emergency.

Remark: In the ULM mode, these commands empty the interpolation buffer once the movement has been stopped (this is not the case in the other modes). Thus, it is not possible to resume the movement.

Syntax:

***IBRK = ipol_group**

***ISTP = ipol_group**

| Parameter | Value | Comment |
|------------|--------|-------------------------|
| ipol_group | 0 or 1 | Interpolated axes group |

6.4.5.2 **ICONT and ICLRB commands**

After a stop with a IBRK or ISTP (if no error present on the moving axis) command, the movement can be either continued or cancelled to enable a new movement to start from this point.

- For the first case, the **ICONT** command (Interpolation **CONTInue**), valid for the PVT, PT and G-code based modes, allows the user to take the current movement back with the current tangential acceleration for G-code based mode, with ***K561** for PVT and PT.
- For the second case, the **ICLRB** command (Interpolation **CLear Buffer**), valid for all interpolation modes, empties the interpolation buffer and prepares the calculator to receive a new movement. This new movement will have for starting point, the point where the movement stopped after the call of the IBRK or ISTP command.

Syntax:

***ICONT = ipol_group**

***ICLRB = ipol_group**

| Parameter | Value | Comment |
|------------|--------|-------------------------|
| ipol_group | 0 or 1 | Interpolated axes group |

6.4.5.3 **ILOCK and IUNLOCK commands**

When an interpolation command is sent, it is firstly treated by the command manager. The command is then stored in the interpolation buffer to be sequentially executed. The processing done at the command manager level can last more than one cycle. So, it is difficult to ensure a quick movement start. To have a quick one, two commands are available.

The **ILOCK** command stops the progress of the commands present in the interpolation buffer but the pre-processing is done and the interpolation buffer fills up. This command is valid for all modes of interpolation.

The **IUNLOCK** command starts the execution of the awaiting commands. This command has a sense and an effect only if the buffer has been previously stopped by the **ILOCK** command. Thanks to this process, as soon as the **IUNLOCK** command is treated and executed, the movement begins (in one cycle). This command is valid for all modes of interpolation.

Syntax:

***ILOCK = ipol_group**

***IUNLOCK = ipol_group**

| Parameter | Value | Comment |
|------------|--------|-------------------------|
| ipol_group | 0 or 1 | Interpolated axes group |

The monitoring ***M531** allows the user to know if Interpolation buffer is locked or not. The depths correspond to the ipol group numbers (depth 0: group 0, depth 1: group1).

| Monitoring | Value | Comment |
|------------|-------|----------------------------------------|
| *M531 | 0 | The interpolation buffer is not locked |
| | 1 | The interpolation buffer is locked |

6.4.5.4 HLT, HLO and HLB commands

The **HLT** and **HLO** command stops the sequences, the interpolated movement with an infinite deceleration and disables the interpolation mode. This command is valid for all interpolation modes. Those two commands have a different effect on the AccurET controller.

The **HLB** command stops the sequences, the interpolated movement with a defined deceleration and disables the interpolation mode at the end of the movement brake. This deceleration is defined by the last programmed deceleration in G-code based mode, the value of *K532 and *K533 parameters in ULM mode and with *K561 in PVT and PT modes.

Syntax:

***HLT[.<axis>]**

***HLO[.<axis>]**

***HLB[.<axis>]**

Remark: The [.<axis>] field is optional.
Refer to [§7.5](#) for more information about the use of the HLT, HLB et HLO commands with the parameter K144.

If the HLO command is also sent to a controller (*HLO.<axis>), the power of this controller will be switched off.

6.4.6 Time commands

6.4.6.1 IWTT command

The **IWTT** (Interpolation **WaiT** Time) command enables the insertion of a waiting time between two movements. The 'waiting_time' parameter is not given with a fixed time unit. It depends on *K526 parameter which includes the power of ten of a second. For example, if *K526 contains the value -3, the user time unit will be the millisecond. This command is valid for all interpolation modes.

Syntax:

***IWTT = ipol_group, waiting_time**

| Parameter | Value | Comment | Units |
|--------------|-------------------------|--------------------------------|----------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| waiting_time | From 0 to time_max (**) | Waiting time between movements | ufti (*) |

(*) Refer to [§6.1.1](#) for more information.

(**) 'Time_max' parameter results from the definition of the time unit and the internal resolution of the UltimET Light.

Example:

On the interpolation group 0 with a time in [ms], a waiting command of 0.5 second will be: *IWTT = 0, 500

6.4.6.2 ICAM command

The **ICAM** (Interpolation **CAM**) command, valid for PVT, PT and G-code based modes, is an alias of the parameter ***K717** which reduces by a certain percentage the kinematic quantities of a movement such as tangential speed, tangential acceleration, tangential deceleration (but not the position to reach). Therefore, the time quantity is increased. Similarly, with the IWTT command, time quantities such as time and lapse of time between two movements are increased in the same proportion. The ICAM command acts directly at the next stop (as well for concatenated and non-concatenated movement). The formulas calculating the new values of these quantities are:

For tangential speed, the formula is:

$$\text{Tangential_speed} \cdot \frac{\text{ICAM}}{100}$$

For tangential acceleration and tangential deceleration, the formula is (similar for deceleration):

$$\text{Tangential_acceleration} \cdot \frac{\text{ICAM}}{100} \cdot \frac{\text{ICAM}}{100}$$

For time quantities, the formula is:

$$\text{Time_quantity} \cdot \frac{100}{\text{ICAM}}$$

Syntax:

***ICAM = <P1> or *K717 = <P1>**

| K | Parameter | <P1> value | Comment | Units |
|-------|-----------|---------------|----------------------------------------------------------------------------------------------|-------|
| *K717 | <P1> | From 1 to 100 | Reduction of the movement's characteristics. The depths correspond to the ipol group numbers | [%] |

Example:

We have a XY system with two orthogonal and linear axes. The interpolation group is already initialized on the system. The user unit for position, speed, acceleration and time are respectively the micrometer, micrometer per second, micrometer per squared second and the millisecond.

```
*ITSPD=0,10000;           // tangential speed of 1 cm/s
*ITACC=0,1000000;         // tangential acceleration of 1 m/s2
*ITDEC=0,1000000;         // tangential deceleration of 1 m/s2
*ITJRT=0,10;              // jerk time of 10 ms
*ILINE=0,10000,0,0,0;     // movement from point (0,0) to point (1cm,0)
```

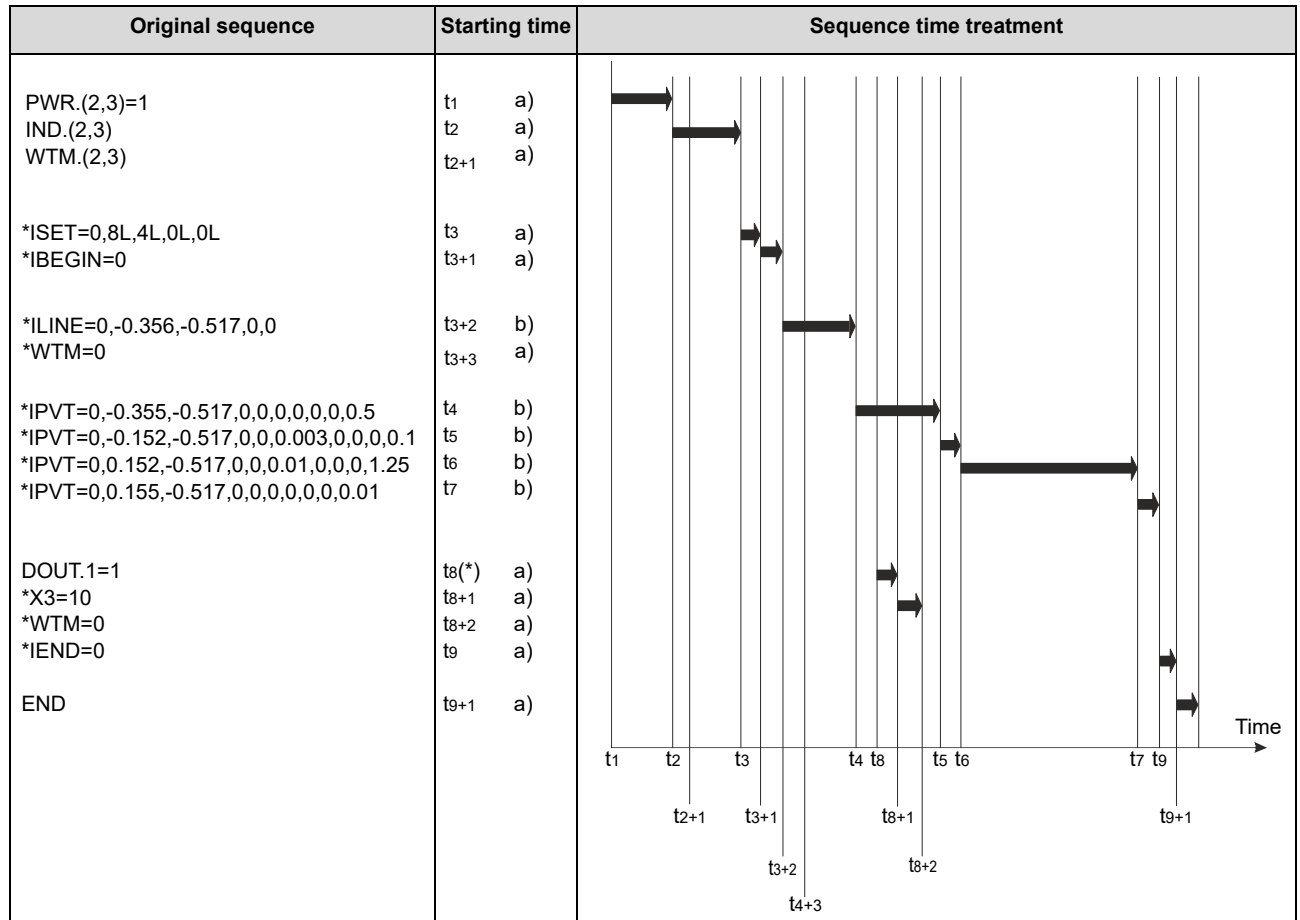
Until then, the movement is done with the kinematic and time values (speed = 1 cm/s, acceleration and deceleration = 1 m/s2 and jerk time = 10 ms).

```
*ICAM=10;                 // global diminution to 10%
*ILINE=0,10000,10000,0,0; // move to the point (1cm,1cm)
*WTM=0;                   // waiting for the end of the movement
```

Remark: The length of the first two movements is identical but the execution time is different.

6.4.7 Example of time treatment of interpolated commands

Example 1:

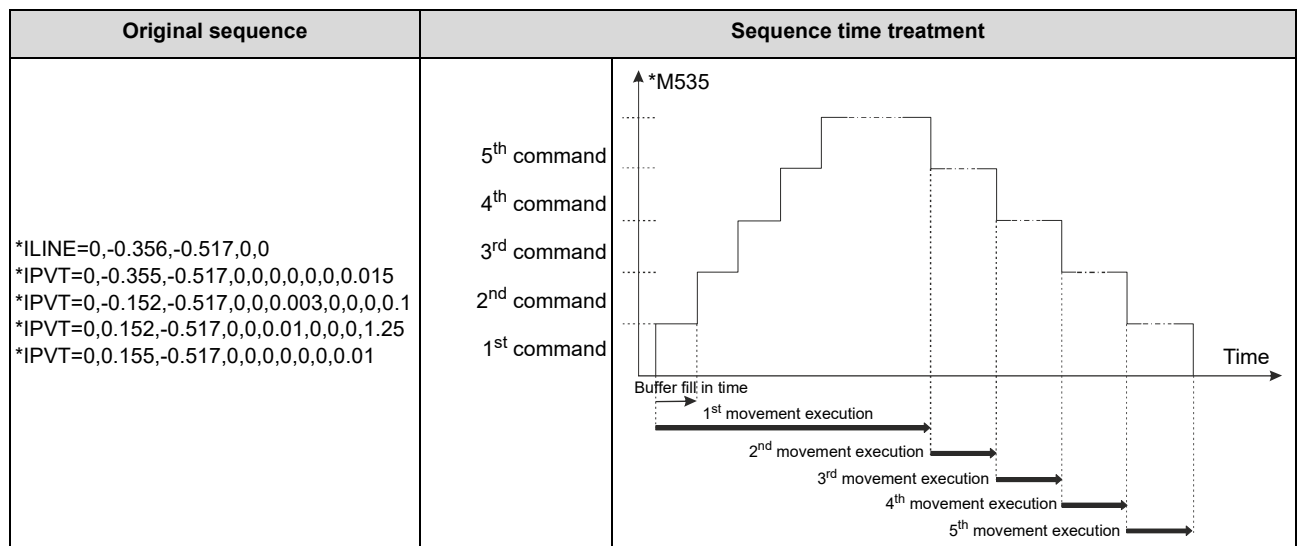


Remark:

- a)=> commands are directly processed as normal command
- b)=> commands are transferred to interpolation buffer and start previous movements are finished.
- tx+n means tx + n command processing cycle
- (*): DOUT command starts when all IPVT commands are pre-processed and stored in the interpolation buffer.

Both buffers (normal and interpolation commands) are processed in parallel that is why some commands are executed before others, even if originally they were placed in a different order.

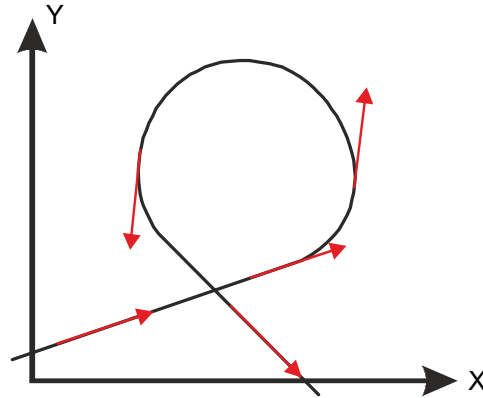
Example 2:



Remark: Buffer fill in time can change according to the UltimET tasks occupation rate. The minimum time is 1.6ms.

6.4.8 Tangential speed

The tangential speed is the speed along the trajectory. This speed can be used for any interpolation move type (G-code based, PVT...) and is accessible when the interpolation mode is active (red arrow on the drawing below).



The monitorings ***M680** and ***MF680** provide the interpolation tangential speed.

| M | Alias | Name | Comment | Unit |
|---------------|----------|----------------------------------------|--------------------------------------------------------------------------------------------------------------------|---------------------|
| *M680 | TANSPD | Current interpolation tangential speed | Gives the current interpolation tangential speed. The depth (0 and 1) correspond to the interpolation group number | UFSI |
| *MF680 | TANSPEED | Current interpolation tangential speed | Gives the current interpolation tangential speed. The depth (0 and 1) correspond to the interpolation group number | ISO (m/s or turn/s) |

By default all interpolated axes contribute to the tangential speed calculation. By using the **ITANSPDMASK** command, the user is able to specify which interpolated axes (X, Y, Z and Θ) contribute to this tangential speed. E.g. a 3D (XYZ) move is done and the user is interested only by an XY tangential speed.

Syntax:

***ITANSPDMASK = ipol_group, interpolatedAxesMask**

| Parameter | Value | Comment |
|----------------------|---------------|----------------------------------------------------------------------------|
| ipol_group | 0 or 1 | Interpolated axes group |
| interpolatedAxesMask | From 0 to 0xF | Mask of interpolated axes contributing to the tangential speed calculation |

This command belongs to the category of commands sent to the interpolation buffer. It is then taken into account at the time it is processed. E.g. the user can start interpolated moves with a tangential speed based on X, Y and Z, and then after some moves, decide that the tangential speed is now restricted to X and Y.

Example: (position unit in micrometers)

```

*ITANSPDMASK= 0, 0x7;           // interpolation group 0, X, Y and Z involved for tangential speed
*iline = 0, 10000, 10000, 10000, 0;
*ITANSPDMASK= 0, 0x3;           // interpolation group 0, X and Y involved for tangential speed
*icw = 0, 10000, 10000, 0, 0;   // full clockwise circle, center at X=0, Y=0
...

```

The current interpolation tangential speed axes mask is accessible through the monitoring ***M681**. This mask is set either by default after ***IBEGIN** command (mask of present interpolated axes) or via ***ITANSPDMASK** command.

| M | Alias | Name | Comment |
|--------------|------------|--------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| *M681 | TANSPDMASK | Current interpolation tangential speed axes mask | Tangential speed is calculated based on axes present in this mask. The depth (0 and 1) correspond to the interpolation group number. |

6.5 'Moving' bits

The 'moving' bits indicate if one of the interpolation groups (ipol0 & ipol1) is executing a movement. This information is given by the monitoring ***M518**. To have the bit equal to 0, the interpolation group must have its current movement finished and the interpolation commands buffer empty. Interpolation status information is also provided in the monitoring ***M519**. Each depth of this register corresponds to an interpolation group.

| Monitoring | Comment |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| *M518 | Interpolation group moving state. Ipol group 0 moving state corresponds to bit 0 (1 = moving), Ipol group 1 moving state corresponds to bit 1. |
| *M519 | Interpolation group status (depth 0=group 0...): 0x1 = active state, 0x2 = moving state and 0x4 = emergency brake state. |

6.6 Marks

The marks are interpolation instructions which synchronize the movements with other tasks. As the other interpolation instructions, the marks are put in the interpolation buffer. When the trajectory calculator meets a mark, it indicates it and goes to the next instruction.

Each mark is associated with a number and possibly an action. The number enables the sequence or the PC to identify the mark. The action, programmable via the command's parameters, enables the association of a selective event as an interruption on the host computer.

Syntax:

***IMARK = ipol_group, mark_number, action, parameter1, parameter2**

| Parameter | Value | Comment |
|-------------|------------------------|-------------------------------------------|
| ipol_group | 0 or 1 | Interpolated axes group |
| mark_number | From 0 to $(2^{31}-1)$ | Number of the mark |
| action | 0-6 | Action to be done |
| parameter1 | - | First parameter associated to the action |
| parameter2 | - | Second parameter associated to the action |

Remark: In ULM movement, the ***IMARK** command is applied in the middle of the transition time.

It is recommended to use only one ***IMARK** command between two move commands. However, more than 5 consecutive ***IMARK** commands is not supported.

6.6.1 Identification of the mark

When the trajectory calculator meets a mark in the buffer of the interpolation commands, it systematically copies the mark's number in the parameter ***K540** at the depth defined by the group of interpolation. Thus, the parameter ***K540** owns the number of the last mark met.

| Parameter | Comment |
|--------------|--------------------------------------------------------------|
| *K540 | Number of the last mark processed by the interpolation group |

6.6.2 Action associated with a mark

The third, fourth and fifth parameters of the **IMARK** command allow the user to specify an action to be done by the trajectory calculator when it meets the mark:

| Operation number | Description |
|------------------|------------------------------------------------------------|
| 0 | Does nothing |
| 1 | Sets the bits of the user status defined by the parameter1 |

| Operation number | Description |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2 | Clears the bits of the user status defined by the parameter1 |
| 3 | Starts a sequence thread (mentioned by parameter2) at the function number defined by parameter1 |
| 4 | Sets an integer user variable whose index and depth are given in parameter1 with the value of parameter2 |
| 5 | Sets / resets digital outputs. The state of the outputs is given in parameter1 (Bit 0-15: set, bit 16-31: reset). It is possible to activate the outputs at the beginning of the next cycle, after a delay defined by parameter2 (step of 30ns and 0 means no delay) |
| 6 | Sets / resets several user status bits. The state of the bits is given in parameter1 (Bit 0-15: set, bit 16-31: reset) |

The parameters *K541 allows the user to reserve digital outputs for the marks:

| Parameter | Comment |
|-----------|-----------------------------|
| *K541 | DOUT reserved for the marks |

Several solutions are possible to use the marks in the sequences:

- To directly read the parameter *K540
- the WTK (WaiT mark) command
- **WTK command**

The **WTK (WaiT for mark)** command allows the user to wait for the execution of a mark by the trajectory calculator.

Syntax:

***WTK = ipol_group, mark_number**

| Parameter | Value | Comment |
|-------------|--------------------------|---------------------------|
| ipol_group | 0 or 1 | Interpolated axes group |
| mark_number | From 0 to ($2^{31}-1$) | Number of the waited mark |

The WTK command constantly checks the parameter ***K540** until it takes the 'mark_id_number' value; the waiting is then finished.

If the parameter *K540 has already the 'mark_id_number' value when the WTK command is executed, the waiting ends immediately. If several marks happen one after the other, the WTK command can miss an intermediate mark. The waiting must be done on the last mark.

Example:

```
*IMARK=0,0,0,0,0;           // group 0, mark 0
*ILINE=0,2000,3000,0,0;     // interpolated movement on the group 0
*IMARK=0,122,0,0,0;        // group 0, mark 122
*ILINE=0,400,6000,0,0;     // interpolated movement on the group 0
*ILINE=0,0,0,0,0;          // interpolated movement on the group 0
*WTK=0,122;                // waits for the mark 122 on the group 0
DOUT.1=1;                  // sets the digital output 1 of the node 1
```

In the above-mentioned example, DOUT1 of the node 1 will be set to 1 when the X and Y axes will have finished the first segment (coord 2000, 3000). X and Y will then continue the movements.

These commands are usable for all interpolation modes.

6.6.2.1 Set / clear bits of the user status

To set / clear bits of the user status, 2 data items is needed:

- the action value saying if the bits will be set or cleared (defined by <action>)
- the bit field of the user status (defined by <parameter1>)

Example:

The user wants to first set and reset bits of the user status (*K177) with the mark number 3. To do so, the command will be as follows:

The initial state of *K177 is 0x8

```
*IMARK=0,3,1,0x30,0;
```

The state of *K177 after the execution of the *IMARK command is 0x38

```
*IMARK=0,3,2,0x28,0;
```

The state of *K177 after the execution of the second *IMARK command is 0x10

6.6.2.2 Start of a sequence

To start a sequence in the UltimET Light, 3 data is needed:

- the action value saying that a jump will be executed (defined by <action>)
- the function number (defined by <parameter1>)
- the thread number used to execute this sequence code (defined by <parameter2>)

Remark: Please refer to [§7.2.8](#) for more information about the threads.

Example

The user wants to send the *IMARK command to start the function number 100 in the thread number 1 and the mark number 5. To do so, the command will be as follows:

```
*IMARK=0,5,3,100,1;
```

6.6.2.3 Setting of an UltimET Light's user variable

To modify a UltimET Light's user variable (X), 4 data is needed:

- the action value saying that a X variable will be modified (defined by <action>)
- the index and the depth of the X variable (defined by <parameter1>)
- the new value of the X variable (defined by <parameter2>)

The depth and the index of the variable X are mixed in a 32-bit word:

```
[31 depth of the X variable 16|15 index of the X variable 0]
```

Example:

The user wants to modify the value of the variable *X10 to 12345, at the depth 2, with the mark number 5. To do so, the command will be as follows:

```
*IMARK=0,5,4,0x0002000A,12345;
```

6.6.2.4 Set / reset a digital output

The user wants to set the DOUT2 and reset the DOUT1 with the mark number 4 and a delay of 30μs (1000 times 30ns step) after the beginning of the next interrupt cycle. To do so, the command will be as follows:

```
*K541=0x3
```

```
*IMARK=0,4,5,0x00010002,1000;
```

6.6.2.5 Set / reset user status bits

The user wants to set the bit# 1 of the user status (*K177) and reset the bit# 3 of the user status (*K177) with the mark number 2. To do so, the command will be as follows:

```
*IMARK=0,2,6,0x00080002,0;
```

6.7 Matrix

To execute circular trajectories in the space (outside the XY plane), a system with a user reference frame and a machine reference frame is used. A reference frame transfer matrix which can be set by means of some functions (translation, rotation, scaling, shearing, restoration of the matrix) is used to pass from a reference frame to the other one.

As described in the ICW, ICCW, ICWR and ICCWR commands, a circle is defined in a user XY plane. To execute a circle in the XZ plane, a 90° rotation of the YZ plane of the user reference frame has to be done. The user command describing the circle in the XY plane is then converted by a circle in the XZ plane at the reference frame level.

Another use of the matrix is the correction of some mechanical defects of a system. During the construction of a mechanical system, it is difficult to ensure an absolute accuracy within a micron. It is possible to correct some defects with a software to increase the accuracy of the system. The mechanical defects linked to the reference frame (non-orthogonal, rotation, scale) can be corrected by the matrix. The irregular defects as a deviation in the rotation on an axis cannot be corrected by the matrix. In these cases, a correction by stage mapping can be used.

6.7.1 Translation function

The **IMTRANS** (Interpolation **M**atrix **T**RANSlation) command allows the user to translate the user reference frame depending on a vector (trans_x, trans_y, trans_z, trans_θ) acting on the four axes.

Syntax:

***IMTRANS = ipol_group, trans_x, trans_y, trans_z, trans_θ**

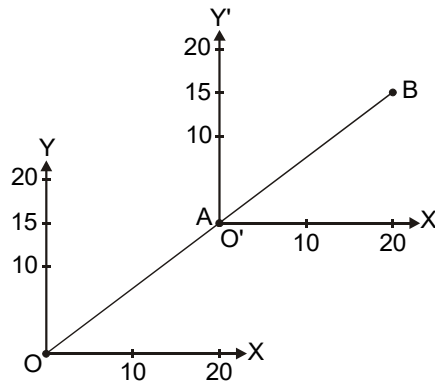
| Parameter | Value | Comment | Units |
|------------|------------------------------------------|--------------------------------------|----------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| trans_x | From -distance_max to +distance_max (**) | x translation of the reference frame | ufpi (*) |
| trans_y | From -distance_max to +distance_max (**) | y translation of the reference frame | ufpi (*) |
| trans_z | From -distance_max to +distance_max (**) | z translation of the reference frame | ufpi (*) |
| trans_θ | From -distance_max to +distance_max (**) | θ translation of the reference frame | ufpi (*) |

(*) and (**) Refer to [§6.1.1](#) for more information.

Example:

In a XY system, the following commands are executed. The interpolated mode is chosen with a tangential speed, a tangential acceleration and a tangential deceleration already known. The user position unit is the micron and the starting point is O (0, 0).

```
*ILINE=0,20000,15000,0,0;      // segment up to point A (X=20mm, Y=15mm)
*IMTRANS=0,20000,15000,0,0;    // translation of the user reference point to the
                                // point (X=20mm, Y=15 mm)
*ILINE=0,20000,15000,0,0;      // segment up to point B (X'=20mm, Y'=15mm)
```



According to the machine (0) reference frame, the user reference frame before translation (1) and the user reference frame after translation (2), the coordinates of the three points are:

| Point \ reference frame | Machine (0) = User (1) | User (2) |
|-------------------------|------------------------|--------------------------|
| O | X = 0, Y = 0 | X' = -20 mm, Y' = -15 mm |
| A | X = 20 mm, Y = 15 mm | X' = 0, Y' = 0 |
| B | X = 40 mm, Y = 30 mm | X' = 20 mm, Y' = 15 mm |

Remark: In the above-mentioned example, the same movement command induces two different machine movements.

6.7.2 Scaling function

The **IMSCALE** (Interpolation **M**atrix **S**CALE) command allows the user to expand or to shrink the user reference frame according to a scale vector (scale_x, scale_y, scale_z, scale_θ) on the four axes.

Syntax:

***IMSCALE = ipol_group, scale_x, scale_y, scale_z, scale_θ**

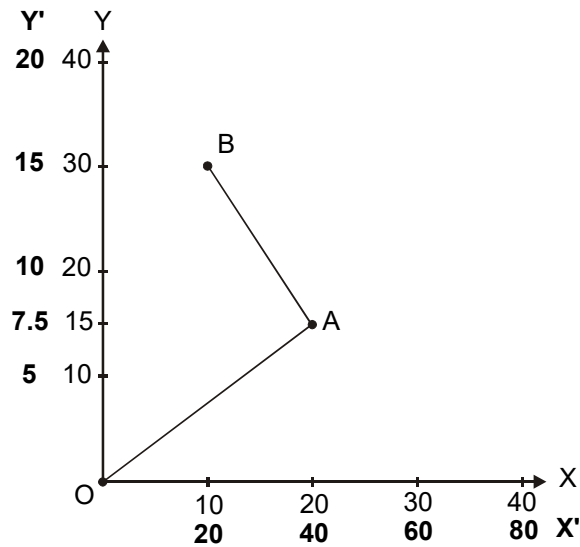
| Parameter | Value | Comment | Units |
|------------|----------------------------|--------------------------------------|-----------------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| scale_x | From -2^{31} to 2^{31} | x coefficient of the reference frame | 1 / million (*) |
| scale_y | From -2^{31} to 2^{31} | y coefficient of the reference frame | 1 / million (*) |
| scale_z | From -2^{31} to 2^{31} | z coefficient of the reference frame | 1 / million (*) |
| scale_θ | From -2^{31} to 2^{31} | θ coefficient of the reference frame | 1 / million (*) |

(*): a scale factor of 1 (scale 1:1) is defined by 1000000. If the value is higher than one million, the scale is expanded. If the value is lower, the scale is shrunk.

Example:

In a XY system, the following commands are executed. The interpolated mode is chosen with a tangential speed, a tangential acceleration and a tangential deceleration already known. The user position unit is the micron and the starting point is O (0, 0).

```
*ILINE=0,20000,15000,0,0;           // segment up to point A (X=20mm,
// Y=15mm)
*IMSCALE=0,500000,2000000,1000000,1000000; // narrow of the X axis by half,
// dilatation of the Y axis by 2
*ILINE=0,20000,15000,0,0;           // segment up to point B (X'=20mm,
// Y'=15mm)
```



According to the machine (0) reference frame, the user reference frame before scale change (1) and the user reference frame after scale change (2), the coordinates of the three points are

| Point \ reference frame | Machine (0) = User (1) | User (2) |
|-------------------------|------------------------|-----------------------|
| O | X = 0, Y = 0 | X = 0, Y = 0 |
| A | X = 20 mm, Y = 15 mm | X = 40 mm, Y = 7.5 mm |
| B | X = 10 mm, Y = 30 mm | X = 20 mm, Y = 15 mm |

6.7.3 Rotation function

The **IMROT** (Interpolation **M**atrix **ROT**ation) command allows the user to make a rotation of the 'plan_rot' plane of the user reference frame according to the 'angle_rot' angle.

Syntax:

***IMROT = ipol_group, plan_rot, angle_rot**

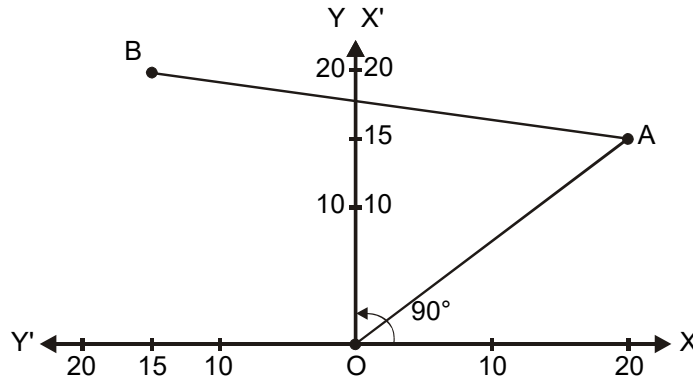
| Parameter | Value | Comment | Units |
|------------|---------------------------------------------|--------------------------------------------------------------------|------------------------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| plan_rot | 0(xy), 1(xz), 2(xθ), 3(yz), 4(yθ), 5(zθ) | - | none |
| angle_rot | From -2^{31} to 2^{31} | Rotation angle of the 'plan_rot' plane of the user reference frame | 1 / million degree (*) |

(*): an angle of one degree is defined by 1000000.

Example:

In a XY system, the following commands are executed. The interpolated mode is chosen with tangential speed, tangential acceleration and tangential deceleration already known. The user position unit is the micron and the starting point is O (0, 0).

```
*ILINE=0,20000,15000,0,0; // segment up to point A (X=20mm, Y=15mm)
*IMROT=0,0,900000000; // 90° rotation of the XY plane
*ILINE=0,20000,15000,0,0; // segment up to point B (X'=20mm, Y'=15mm)
```



According to the machine (0) reference frame, the user reference frame before rotation (1) and the user reference frame after rotation (2), the coordinates of the three points are:

| Point \ reference frame | Machine (0) = User (1) | User (2) |
|-------------------------|------------------------|-----------------------|
| O | X = 0, Y = 0 | X = 0, Y = 0 |
| A | X = 20 mm, Y = 15 mm | X = 15 mm, Y = -20 mm |
| B | X = -15 mm, Y = 20 mm | X = 20 mm, Y = 15 mm |

6.7.4 Shearing function

The **IMSHEAR** (Interpolation **M**atrix **S**HEARing) command allows the user to shear an axis of the reference frame according to the other axes. It is the case when two or more axes are not orthogonal.

Syntax:

***IMSHEAR = ipol_group, sheared_axis, shear_1, shear_2, shear_3**

| Parameter | Value | Comment | Units |
|--------------|----------------------------------------|----------------------------------|------------------|
| ipol_group | 0 or 1 | Interpolated axes group | - |
| sheared_axis | 0(x), 1(y), 2(z) and 3(θ) (*) | Sheared axis | none |
| shear_1 | From -2^{31} to 2^{31} | Shearing according to the axis 1 | 1 / million (**) |
| shear_2 | From -2^{31} to 2^{31} | Shearing according to the axis 2 | 1 / million (**) |
| shear_3 | From -2^{31} to 2^{31} | Shearing according to the axis 3 | 1 / million (**) |

(*): when the x axis is sheared (sheared_axis equal to 0), the three other axes can be affected. The influence on these ones will be defined in 'shear_1' for the y axis, in 'shear_2' for the z axis and in 'shear_3' for the θ axis.

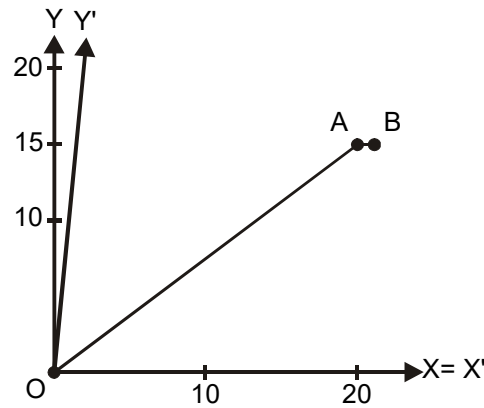
(**): the influence of a sheared axis on the other axes is given in 1 / million. If the y axis is sheared and affects the positioning on the x axis of a factor of 0.1 while the other two axes are not influenced, the command line is:

```
*IMSHEAR=0,1,100000,0,0; // interpolation group 0, shearing of y and influence on x of 10% of y
```

Example:

In a XY system, the following commands are executed. The interpolated mode is chosen with a tangential speed, a tangential acceleration and a tangential deceleration already known. The user position unit is the micron and the starting point is O (0, 0).

```
*ILINE=0,20000,15000,0,0; // segment up to point A (X=20mm, Y=15mm)
*IMSHEAR=0,1,100000,0,0; // shearing of the y axis according to the axis x
// in a ratio of 0.1
*ILINE=0,20000,15000,0,0; // segment up to point B (X'=20mm, Y'=15mm)
```



According to the machine (0) reference frame, the user reference frame before shearing (1) and the user reference frame after shearing (2), the coordinates of the three points are:

| Point \ reference frame | Machine (0) = User (1) | User (2) |
|-------------------------|------------------------|------------------------|
| O | X = 0, Y = 0 | X = 0, Y = 0 |
| A | X = 20 mm, Y = 15 mm | X = 18.5 mm, Y = 15 mm |
| B | X = 21.5 mm, Y = 15 mm | X = 20 mm, Y = 15 mm |

6.7.5 Matrix restore

The **IMRES** (Interpolation **M**atrix **RE**store) command allows the user to restore the default matrix of the selected interpolation group. The effect of the 4 other matrix functions is then cancelled which means the initial user reference is taken into account. The user reference frame coordinates are kept and the matrix translation vector updated in order to maintain the machine reference frame coordinates.

Syntax:

***IMRES = ipol_group**

| Parameter | Value | Comment |
|------------|--------|-------------------------|
| ipol_group | 0 or 1 | Interpolated axes group |

6.8 Conversion factor

The UltimET Light works at the same resolution than the interpolated axis having the biggest resolution. Before being sent to the TransnET, the position references are multiplied by a conversion factor specific to each axis in order to be at the controllers level (upi). This operation converts the UltimET Light increments references into increments specific to the corresponding axis.

To work with a rotary and a linear axis, a correspondence between each of them has to be done. With this standard setting, one turn (rotary motor) is equivalent to 1m (on linear motor).

The user has the possibility to modify this correspondence. It is then possible to choose any ratio between the different axes (linear or rotary). To do so, the parameters ***K580**, ***K581** and ***K582** are needed.

| Parameter | Comment |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| *K580 | User conversion factor to fix a ratio between interpolated axes: activation parameter (0 = disabled, 1 = enabled, depth 0 = group 0, depth 1 = group 1) |
| *K581 | User conversion factor to fix a ratio between interpolated axes: numerator (depths correspond to interpolated axes 0=X,1=Y,...) |
| *K582 | User conversion factor to fix a ratio between interpolated axes: denominator (depths correspond to interpolated axes 0=X,1=Y,...) |

If ***K580 = 1**, the factor of each axes is calculated as follows:

```
user_factor_X_axis = *K581:0 / *K582:0  
user_factor_Y_axis = *K581:1 / *K582:1  
user_factor_Z_axis = *K581:2 / *K582:2  
user_factor_θ_axis = *K581:3 / *K582:3
```

Remark: if *K580 = 1 (activated), do not forget to set *K581 and *K582 to 1 for the axes not used.

Example:

Here is an example with two motors (one linear and one rotary) driven by two AccurET. (Refer to the '**Operation and Software Manual**' for more information about the parameters and the conversion formulas).

The rotary motor has an analog encoder with 5000 lines/turn ($K77 = 3$) and the linear motor has a scale with a period of $20\mu\text{m}$ ($K77 = 4$).

If the user wants to have 1 turn equivalent to 12.3 mm, the ratio is as follows:

$$1 \text{ turn} = 0.0123 \text{ meter} \quad \leftrightarrow \quad 1 \text{ meter} \cong 81.300813... \text{ turns}$$

In standard application, the number of turns that may be executed by a rotary motor is bigger than the number of meters done by a linear motor. Thus, in the turn/meter correspondence, the linear axis is restrictive for the rotary axis. The user should modify the ratio of *K581/*K582 of the rotary axis.

*K581 = 10000 and *K582 = 123 for the rotary motor

*K581 = 1 and *K582 = 1 for the linear motor

Then, $1 \text{ meter} = 10000 / 123 = *K581 / *K582$

Chapter E: Programming

7. Programming of the controller

7.1 Introduction

When required by the application, it is possible to write a program that will be executed by the controller itself. This program is called 'sequence' because it is a list of instructions that are executed sequentially by the controller. These instructions can start movements, check the I/O state, change the controller registers, etc.

A sequence is written in a specific ETEL language and can be stored in a text file on your PC. In order to be executed, the sequence has to be compiled, downloaded and stored in the controller. This is done thanks to ComET.

The size of the memory dedicated to the sequence is:

- Compiled code: 256 KB
- Global data: 96 KB
- Source code: 256 KB

Any size exceeding is notified by the compiler. For the size of the execution stacks, please refer to [§7.7.2](#).

The execution of the sequence starts when a JMP command is sent to the controller. A sequence can also be executed at boot time, if the controller has been configured to automatically start the sequence.

A sequence can contain several routines. Each routine can be executed separately or can be combined to build a more complex program. For example, the controller can host a sequence with 5 routines, each one handling one of the 5 specific movements or actions needed by the application. At any time, the host PC can perform the needed movement or action by starting one of these routines on the controller. The sequence is also convenient to execute supervision tasks or to customize tasks (the homing process, for example).

Before downloading a new firmware version, please refer to [§3.1.3.4](#) (for PCI and PCIe) or [§3.3.4.3](#) (for TCP/IP) for the sequence management.

The parameters ***K297** allows the users to select the mask for active thread information on M63 status (bit#31) (refer to [§4.14.1](#)).

| K | Name | Comment |
|--------------|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *K297 | Mask for active thread | Mask for active thread information on M63 status (bit#1 for thread1, bit#2 for thread2 and bit#3 for thread3). It is a bit field, then the maximum value is 14. |

- *K297=0 by default. It clears bit#31 of M63.
- Bit#1 of *K297 allows the user to show thread 1 running on bit#31 of M63.
- Bit#2 of *K297 allows the user to show thread 2 running on bit#31 of M63.
- Bit#3 of *K297 allows the user to show thread 3 running on bit#31 of M63.

7.2 Basic concepts

7.2.1 Source file

A sequence is a text file describing the instruction sequence thanks to a specific ETEL language. The sequence is stored in a .cseq file. This file extension is coming from 'compiled sequence' and it is also used to differentiate the AccurET and UltimET sequence from the .seq file used by the previous products. The file can be edited with a standard text editor (Notepad, Notepad++, ...) or by ComET. A sequence must always be stored in a single file. There is no possibility to handle multi-file source code.

7.2.2 Comments

In the sequence, each line starting with '/' is a comment and is not taken into account by the compiler. The same is for any text placed between a "/" and a "*".

Lines starting with '#' are special comments generated and managed by various ETEL tools. Please, do not use these syntax for your own comments.

7.2.3 Header

Each sequence containing ISO values must have a header defining the units available in the sequence (if the sequence does not contains ISO value, the header is not necessary). The header is composed of few lines starting with '#'. These lines can be generated by ComET by choosing 'Insert ISO header' and/or 'Insert version header' in the 'Edit' menu. The sequence developer can adapt it with the wished units by modifying them in ComET. To do so, click on 'File' and 'Preferences'.

7.2.4 Register and command

Each of the functions available in the controller, is managed by a set of registers (KF80, K177, ...) and commands (MVE, STA, PWR, IND, ...).

The registers and command can be reached and executed by the ComET Terminal or by the application running on the host PC. In the same way, these registers and commands can be reached and executed by the sequence.

The ComET Terminal and the sequence have a very similar syntax. Any command available on the ComET Terminal can be used with the same syntax on the sequence. There is few exceptions that will be developed later in this manual.

7.2.5 Instruction delimiter

A semicolon (";") must be placed after each instruction (each expression statement). So, on a single text line it is possible to place several instructions separated by semicolons.

An instruction can be written on more than one text line. The 'Carriage return' (CR) or 'Line feed' (LF) characters ending each text line have no special meaning from the language point of view.

7.2.6 Immediate values

Each time a number is written in the code, the type of this number must be specified. There are 5 different types giving 5 different syntaxes.

| Type | Syntax example | Description |
|-----------------------------|--------------------------|-----------------------------------------------------------------------------|
| Integer (32-bit values) | 123 or -200 or 0x12AB | Simple number, possibly preceded by the '0x' prefix for hexadecimal values. |
| Long (Integer 64-bit value) | 123L or -200L or 0x12ABL | As an integer 32-bit value followed by the 'L' suffix. |
| Float (32-bit values) | 123F or 1.2F or -3.5F | Standard engineering notation followed by the 'F' suffix. |
| Double (float 40-bit value) | 123D or 1.2D or -3.5D | Standard engineering notation followed by the 'D' suffix. |
| ISO value | 123.0 or 1.2 or -3.5 | Standard engineering notation without any suffix. |

The ISO values are values representing physical quantities. The units applied to these values are the basic ones defined by the ISO standard (m, s, A, kg, ...) except when a specific unit is defined in the sequence header (refer to [§7.2.3](#)). Refer to [§4.2.1.1](#) for more information about the registers.

ISO value can be used only when the physical unit is implicitly known. For example, when the KL212 (or ACC) register is set, the compiler knows implicitly that KL212 stores an acceleration and that the ISO unit is the meter/s². Thus, it is possible to write:

```
KL212 = 0.02;
```

Please note that the conversion is done during the compilation of the code. No conversion is allowed at run time. ISO value can be used as an argument of the iconv(), lconv(), fconv() and dconv() functions. Refer to [§7.3.9](#) for more information.

7.2.7 C preprocessor define

It is possible to declare C preprocessor define in sequences.

Example:

```
#define AXIS_X 1
#define AXIS_Y 2

void func10(void)
{
    RST.AXIS_X;
    PWR.AXIS_X = 1;
}
```

However, there are some restrictions, comparing to C preprocessor define.

- It is not possible to declare a #define from another #define.
- It is not allowed to use an identifier similar to an ETEL specific command or register

Example:

```
#define AXIS_X 1
#define AXIS_Z AXIS_X + 1           // This is not allowed!
#define M0 0                       // This is not allowed!
#define SAV 4                      // This is not allowed!
```

7.2.8 Thread

It is possible to run several routines at the same time to organize your code and program separated tasks. This is similar to having several processors, each one executing a routine or a portion of the whole sequence. In practice, there is only one processor that operates as several virtual processors. Each virtual processor is called a 'thread'. The UltimET controller has 3 threads per axis. The monitoring **M97** allows the user to know which thread is currently active:

| M | Name | Comment |
|------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *M97 | Sequence thread | Mask of the sequence threads currently active (bit #0 is not used, bit #1=1 if the thread 1 is active, bit #2=1 if the thread 2 is active and bit #3=1 if the thread 3 is active). |

Basically, only one thread is needed to execute the sequence, but with several threads, complex programming (multi-tasking) is possible.

The code of a thread can be shared. Actually, It is also possible to have two threads executing the same routines. For example, a routine computing the average on 2 values can be written. Two threads can perform very different tasks in which they need to compute an average. So, both threads will call this routine (refer to [§7.4](#) for more information).

7.2.9 User variables X

The user variables X are reserved to the user program. Each thread owns 512 variables.

Syntax:

X<number>[:<thread>]

| Parameter | Value | Comment |
|-----------|----------|----------------------|
| number | 0 to 511 | User variable number |

| Parameter | Value | Comment |
|-----------|--------|-------------------------------------------------------------------------------------------------|
| thread | 1 or 3 | 0: X user variable of the current thread 1, 2 or 3: User variable X of the designated thread |

Remark: The [:<thread>] field is optional.

When the thread number is not specified or is equal to 0, it refers to the current thread. It enables the writing of code which can be simultaneously executed by several threads, each thread using its own X variables.

Example:

```
*X1=44;
```

If it is executed by the thread 1, this instruction will set the value 44 in *X1:1.

If it is executed by the thread 2, this instruction will set the value 44 in *X1:2.

Etc...

This technique is useful to write subroutine for general use.

The user variables X are also useful to pass information or synchronize two threads. The threads can also read and write in the variables X of another thread. To do so, a thread number must be specified. For example, the instruction *X1:2 = 22 will set the value 22 in the variable X1 of the thread number 2 whatever the thread executing the instruction.

The user variables X of each thread can also be read and written by the PC via the PCI/PCle or TCP/IP port. There is only one difference in the syntax: when the thread is not specified (*X1) or is equal to 0 (*X1:0), it refers to the variables X of the thread 1 because the PCI/PCle or TCP/IP port does not have its own variables set.

7.2.10 Advanced management of the X variables in a multi-thread configuration

Before using this functionality, the user must have read and understood the multi-thread management and the use of the local variables X (refer to [§7.2.8](#) and [§7.2.9](#) for more information).

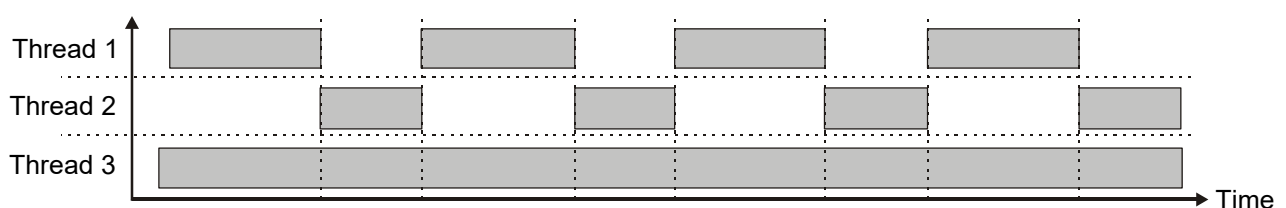
Here is an application example doing interpolated movements, which may be stopped anytime by a security process and using the UltimET Light's resources as follows:

- thread 1: generation of the interpolated movements
- thread 2: calculation of a set of parameters for the next block of movements
- thread 3: continuous security routine

```
void func100(void)
{
    *ILINE=0,0,0,0,0;           // moves to X=0, Y=0, Z=0 and Theta=0
    *ILINE=0,*X0,*X1,0,0;       // moves to X=*X0 and Y=*X1
    *END;
}
```

This small sequence is used for all the trajectories. However, the X variables must be recalculated which means that the user has to wait for the end of the movement. As long as the thread 1 generates the trajectory, the thread 2 waits for the end of the movement (thread 1). Once the movement has been executed, the thread 2 recalculates the X variables of the thread 1. The thread 3 is continuously running for security.

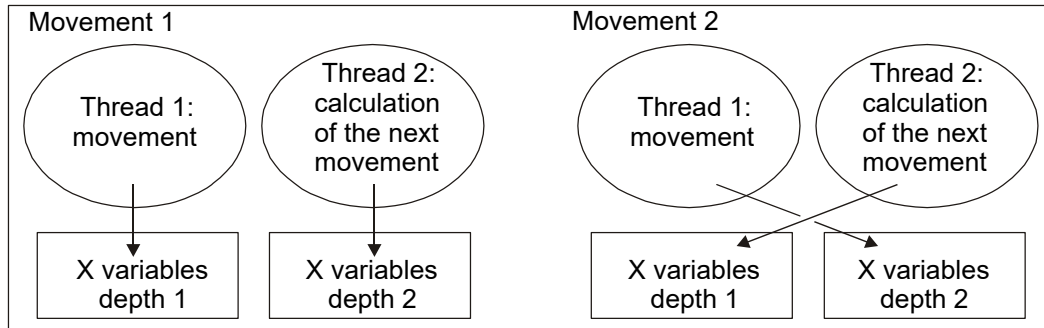
Here is the time division for each thread:



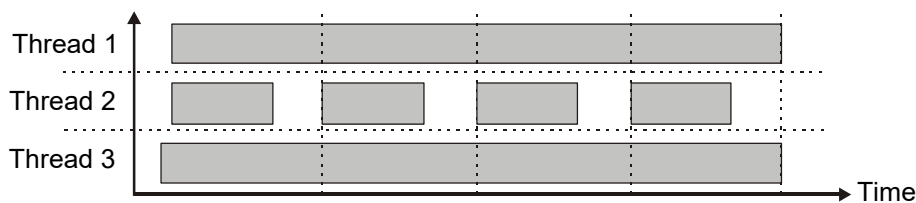
The tasks of the threads 1 and 2 are not done in parallel but one after the other. The purpose of this

management is to reduce this delay.

To execute the two tasks in parallel, the thread 2 must use its own X variables (depth 2) while the thread 1 uses its own X variables (depth 1). Once finished, some commands can be executed, allowing the use of the X variables at depth 1 as local variables for the thread 2 and the use of the X variables at depth 2 as local variables for the thread 1. A swap of the X variables is performed.



Thanks to this function, both tasks can be executed in parallel. Here is the new time division for each thread:



The ***K143** parameter has been developed to enable this functionality. It allows the user to configure the X locale variables which will be used by a thread. Each depth of the ***K143** parameter corresponds to a thread (depth 1 for thread 1, depth 2 for thread 2 and depth 3 for thread 3). The values of the ***K143** parameter correspond to the depths of the X variables locally used by the threads (e.g. ***K143:1=2** means that the thread 1 uses the depth 2 of the X variables as its local X variables).

Here is a short sequence executed alternatively by each thread:

```
void func10(void)
{
    *X0+=1;
    *END=0;
}
```

Here is the effect, on the sequence, of the following sets of the ***K143** parameter:

- By default, the ***K143** parameter is equal to 0 at each depth. In this case and in order to keep the compatibility with the previous UltimET Light's firmware versions, the standard functioning mode is applied:

| *K143:depth = value | Command | Modified variable |
|----------------------------|------------------|--------------------------|
| *K143:1=0 | *JMP=10,1 | *X0:1+=1 |
| *K143:2=0 | *JMP=10,2 | *X0:2+=1 |
| *K143:3=0 | *JMP=10,3 | *X0:3+=1 |

Remark: The modified variables would have been the same if the values of ***K143** parameter were equal to 1, 2 and 3 at the depths 1, 2 and 3.

- K143** is now set as follows:

```
*K143:1=3;      // the depth 3 of the X variables is locally used by the thread 1
*K143:2=1;      // the depth 1 of the X variables is locally used by the thread 2
*K143:3=2;      // the depth 2 of the X variables is locally used by the thread 3
```

If the thread 1 executes the above-mentioned sequence (*JMP=10,1), the *X0:3 variable will be incremented and the other depths (1 and 2) of this variable will not change. If the thread 2 executes this sequence (*JMP=10,2), the *X0:1 will be incremented and so on.

| *K143:depth = value | Command | Modified variable |
|---------------------|-----------|-------------------|
| *K143:1 = 3 | *JMP=10,1 | *X0:3+=1 |
| *K143:2 = 1 | *JMP=10,2 | *X0:1+=1 |
| *K143:3 = 2 | *JMP=10,3 | *X0:2+=1 |

- K143 is now set as follows:

```
*K143:1=2;           // the depth 2 of the X variables is locally used by the thread 1
*K143:2=2;           // the depth 2 of the X variables is locally used by the thread 2
*K143:3=2;           // the depth 2 of the X variables is locally used by the thread 3
```

If the thread 1 executes the above-mentioned sequence (*JMP=10,1), the *X0:2 variable will be incremented. If the thread 2 executes this sequence (*JMP=10,2), the *X0:2 will also be incremented and if the thread 3 executes this sequence (*JMP=10,3), the *X0:2 will be incremented too.

| *K143:depth = value | Command | Modified variable |
|---------------------|-----------|-------------------|
| *K143:1 = 2 | *JMP=10,1 | *X0:2+=1 |
| *K143:2 = 2 | *JMP=10,2 | *X0:2+=1 |
| *K143:3 = 2 | *JMP=10,3 | *X0:2+=1 |

7.3 Structured code

When the applications become complex, it is important to have a language helping the developer to organize his code. A good structure often results in a code more reliable and easier to maintain and reuse. The C language is very popular in the industry and has inspired lots of other languages being used in the industry, as Java, C++, C#, etc..

To improve the sequence language and introduce a structured syntax, a solution based on the industry standards has been built. ETEL's sequences integrate the basic functionalities of the C language in what is called the 'structured code'. The developer being familiar with the C language will quickly feel comfortable with the ETEL sequences. However, basic knowledge in the C language are recommended to enjoy the advantages of the structured code.

The ETEL sequence language has to be seen as a subset of the C language. It includes all the features of the C having a benefit for embedded motion control applications. Pointers, for example, are not available in sequences, because they are not suitable for embedded sequences and can easily decrease reliability. On the other hand, some specific syntax has been added to give a direct access to all ETEL commands and registers (PWR, WTM, ...) and, in this way, simplify the development of motion control applications. The combination of C language basis with the ETEL commands and registers results in a very powerful language, easy to learn and perfectly suitable for the development of motion control application.

7.3.1 Identifiers

An identifier is a name used to identify functions, function arguments and variables. The name is a case sensitive character string, containing any of the following character:

- Alphabetic characters (from 'a' to 'z' and from 'A' to 'Z')
- Digits (from 0 to 9)
- Underline sign ('_')

An identifier always starts with an alphabetic character.

Remark: It is not possible to declare a variable with a name similar to any register name.

7.3.2 Types

The available types are integer, long, float and double. These types are used to define variables, function arguments and function returns.

| Type | Description |
|--------|-------------------------------|
| int | Integer value, 32-bit, signed |
| long | Integer value, 64-bit, signed |
| float | Floating-point value, 32-bit |
| double | Floating-point value, 64-bit |

Remark: At the controller level, the 'Double' type value are truncated to 40 bits.

Type conversion is automatically done, except on function and command calls.

Example:

```
float fct(int v)
{
    return 1;                // Not allowed
    return 1.3F;
}
void func10(void)
{
    X0.0 = 1.5F;              // Is accepted, X0.0 will be equal to 1
    X0.0 = 1.6D;              // Is accepted, X0.0 will be equal to 1
    X0.0 = 0xFFFFFFFFFFFFFFFFL; // Is accepted, X0.0 will be equal to -1
    X0.0 = fct(1);            // Is accepted, X0.0 will be equal to 1
    X0.0 = fct(1.0F);         // Not allowed
}
```

7.3.3 Global variables

The global variables are variables accessible from any function present in the sequence. They must be defined before any function definition.

The syntax used to define a global variable is the following (same as in C):

<type> <variable name>;

It is possible to specify a value which the variable is initialized with:

<type> <variable name> = <immediate value>;

Example:

```
int loop_counter;
float speed = 1.2F;
```

The <type> is one of the 4 types described in [§7.3.2](#).

The <variable name> is an identifier described in [§7.3.1](#).

The <immediate value> must comply with [§7.2.6](#).

Global variables can be used in expressions (refer to [§7.3.7](#)).

Caution: When a global variables is initialized with a value, the initialization takes place only when the sequence is downloaded or when the sequence is restored from the flash (at boot time or with the RES command). When a sequence starts, the values stored in the global variable are not modified (are not initialized). They keep the value set by any sequence executed before.

7.3.4 Registers

In the sequence code, it is possible to have a direct access to all existing registers (X, M, K, XL, KL, ...). These registers work as global variables but they do not need to be declared.

For example, is it possible to write expression statements involving registers:

```
X1 = M6 + X4 / 2;
```

This possibility is not present in the standard C language.

7.3.5 Functions

7.3.5.1 Function definition

A function is a piece of code (a routine) having a name and, possibly, some arguments and a return value. All code (all routines) must be placed in a function. Functions are located after the definition of global variables.

Here is the syntax of the definition of a function:

```
<return type><function name>(<arguments>)  
{  
    <function body>  
}
```

The <return type> can be one of the 4 types described in [§7.3.2](#) or 'void' when there is no return value.

The <function name> is an identifier (refer to [§7.3.1](#)).

The <arguments> are a list of coma separated elements, each one having a type and a name. The type is conformed to [§7.3.2](#) and the name is an identifier conformed to [§7.3.1](#). If no argument is needed, the arguments must be replaced by the 'void' keyword.

Example:

```
void my_routine_to_perform_mouvements(void)  
{  
    // routine code  
}  
  
int get_step_count(void)  
{  
    // routine code  
}  
  
int make_average(int value1, int value2)  
{  
    // routine code  
}
```

When a <return type> is defined, the function returns a value. Inside the function body, the return statement must be used to specify which value has to be returned (refer to [§7.3.8.10](#)).

The <function body> contains the definition of the local variables, followed by a list of statements. Local variables are defined in the same way as global variables (refer to [§7.3.3](#)).

Example:

```
void move_to_park_position(void)  
{  
    int actual_pos;                // variable storing the actual position  
    int target_pos = 10000;        // variable storing the target position  
    ...  
    // place the list of statements here  
}
```

Local variables are available inside the function body only. Refer to [§7.3.8](#) for informations about the statements.

7.3.5.2 Function call

Functions can be called from within an expression.

Syntax:

<function name>(<argument 1>, <argument 2>, ...)

<argument 1>, <argument 2>, etc. can be immediate values, variables, registers or any other expressions.

When a function is called, the function body is executed. If a value is returned, this value takes the place of the function inside the expression.

Example:

```
foo = 2 * average(X1, 2 * bar, abs(X2)) + 1
```

In this expression 'average' is a function. It is called with 3 arguments. The first is 'X1', the second is '2 * bar' and the third is 'abs(X2)'. The result of this function (the returned value) will then be multiplied by 2 and incremented by 1.

7.3.5.3 Interface functions

Some functions can be called from within the sequence itself, and some from the external application like ComET or another EDI-based host PC application.

Functions called from within the sequence itself do not have any naming or declaration constraints other than the specifications above.

Functions that are meant to be called from outside the sequence, must comply with the following rules:

- They must be named **func<n>** where n is any integer from 0 to 1023.
- They cannot return any values nor accept any parameters.

As a result their prototype must look like: **void func<n>(void)**

7.3.5.4 Function declaration

When a call to a function is made, this function must be known. In the source file, this means that the definition of a function should precede its call.

Example:

```
int average(int a, int b)
{
    return (a + b) / 2;
}

void func40(void)
{
    *X1 = average(M7.0, M7.1);    // Call a function defined above => OK
    *X2 = max(M7.0, M7.1);        // Call a function defined below => NOT
                                // ALLOWED!
}

// This function is not yet known because its definition is done later in the file
int max(int a, int b)
{
    if (a > b)
        return a;
    return b;
}
```

In this example, it is quite simple to define the 'max' function before 'func40', avoiding any problem. But depending on the complexity of the code, defining the functions in the right order can result in a headache.

This problem, well known in the C language, can be solved by declaring the functions at the beginning of the source file.

Example:

```
int max(int a, int b);           // This is a function declaration. It just tells
                                // that the function exists, but the whole
                                // definition of the function will be done
                                // later.
int average(int a, int b)       // Definition of the 'average' function.
{
    return (a + b) / 2;
}
void func40(void)               // Definition of the 'func40' function.
{
    *X1 = average(M7.0, M7.1); // Call a function defined above => OK
    *X2 = max(M7.0, M7.1);     // Call a function defined below but declared
                                // above => OK
                                // The 'max' function is known here because of
                                // the declaration above.
}
int max(int a, int b)           // This is the definition of the function
                                // already declared at the beginning of the
                                // file.
{
    if (a > b)
        return a;
    return b;
}
```

Please, do not mix the terms 'declaration' and 'definition'. Function declarations must be located before any function definition and after any definition of global variables.

7.3.6 Operators

The operators define how to combine together the various elements of an expression. A typical example is the sum ('+' operator). It allows the user to add two values. For the operator point of view, the elements on which the operator acts are called operand. The operands can be immediate values, variables or registers.

Example:

| | |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| X1 + 12 | with: 'X1' is the operand (immediate value) '+' is the operator (addition) '12' is the operand (register) |
| foo / 13.4F | with: 'foo' is the operand (variable) '/' is the operator (division) '13.4F' is the operand (immediate value) |

The operand can also be an expression that, once executed, results in a value.

Example:

| | |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|
| X1 + (12 * max(X2, 20)) | with: 'X1' is the operand (register) '+' is the operator (division) '(12 * max(X2, 20))' is the operand (expression) |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------|

Each operator performs an operation on its operands. This normally results in a value which takes the place of the operator and operands in the expression. Each operator has a priority (refer to [§7.3.7](#)).

7.3.6.1 Unary operators

Unary operators are operators acting on a single operand. Here is the syntax:

<operator><operand>

The <operand> can be an immediate value, a variable, a register or any expression resulting in a value.

Example:

-X1 with: '-' is the operator
 'X1' is the operand

Here is the list of unary operators:

| Operator | Priority | Description |
|----------|----------|--------------------------------|
| - | 12 | Sign <right operand> |
| ~ | 12 | Bitwise toggle <right operand> |

7.3.6.2 Binary operators

Binary operators are operators acting on two operands. The typical example is the sum ('+' operator). In an expression, the operator is located between two operands. Here is the syntax:

<operand> <operator> <operand>

The <operand> can be an immediate value, a variable, a register or any expression resulting in a value.

Example:

X1 + 12

Here is the list of binary operators:

| Operator | Priority | Description |
|----------|----------|---------------------------------------------|
| + | 10 | Increment <left operand> by <right operand> |
| - | 10 | Decrement <left operand> by <right operand> |
| * | 11 | Multiply <left operand> by <right operand> |
| / | 11 | Divide <left operand> by <right operand> |
| % | 11 | <left operand> modulo <right operand> |
| & | 6 | <left operand> bitwise and <right operand> |
| | 4 | <left operand> bitwise or <right operand> |
| << | 9 | <left operand> left shift <right operand> |
| >> | 9 | <left operand> right shift <right operand> |
| ^ | 5 | <left operand> XOR <right operand> |

7.3.6.3 Boolean operators

Boolean operators are binary operators which result is true or false. They are often used to compare two operands.

Example:

X1.0 > 12

Here, the '>' operator compares 'X1.0' and '12'. The result of this comparison is not a value. It can only be true or false.

Here is a list of all binary boolean operators:

| Operator | Priority | Description |
|----------|----------|-------------------------------------------------|
| == | 7 | <left operand> test equal to <right operand> |
| != | 7 | <left operand> different <right operand> |
| < | 8 | <left operand> less than <right operand> |
| > | 8 | <left operand> greater than <right operand> |
| >= | 8 | <left operand> greater or equal <right operand> |
| <= | 8 | <left operand> less or equal <right operand> |
| && | 3 | <left operand> logical and <right operand> |
| | 2 | <left operand> logical or <right operand> |

Boolean operators are allowed only in boolean expressions.

7.3.6.4 Assignment operators

All operators described above are performing operations on their operand without affecting the operand itself. Once the operation is performed, the resulting value takes the place of the operator and operands in the expression.

Assignment operators are affecting their left operand. The typical example in the assignment operator '='. When we write, X1 = 23, we expect to change the value stored in the register X1.

The syntax is similar to the one with a binary operator:

<left operand> <assignment operator> <right operand>

The <left operand> can be a variable or a register only!

The <right operand> can be an immediate value, a variable, a register or any expression resulting in a value.

Here is the list of assignment operators:

| Operator | Priority | Description |
|----------|----------|-----------------------------------------------------------|
| = | 1 | Assign the value of <right operand> to the <left operand> |
| += | 1 | Increment <left operand> by <right operand> |
| -= | 1 | Decrement <left operand> by <right operand> |
| *= | 1 | Multiply <left operand> by <right operand> |
| /= | 1 | Divide <left operand> by <right operand> |
| %= | 1 | <left operand> = <left operand> modulo <right operand> |
| <<= | 1 | Left shift <left operand> by <right operand> |
| >>= | 1 | Right shift <left operand> by <right operand> |
| &= | 1 | AND <left operand> with <right operand> |
| = | 1 | OR <left operand> with <right operand> |
| ^= | 1 | XOR <left operand> with <right operand> |

Warning: The atomicity of these operators is not guaranteed. Their execution requires several instructions that may be interrupted. Therefore, if several threads/applications modify similar registers, corruption of data is possible. The command CH_BIT_REG32 could be a safer solution for bitwise operations.

7.3.6.5 Increment and decrement operators

To increment or decrement a register or a variable, the '++' and '--' operators can be used. These operators are unary assignment operators. Their operand can be a variable or a register only.

Example:

```
X1.0++;           // Increment X1.0 by one (same as X1.0 = X1.0 + 1;)
X2.0--;           // Decrement X2.0 by one (same as X2.0 = X2.0 - 1;)
foo++;            // Increment the 'foo' variable by one (same as foo = foo + 1;)
bar--;            // Decrement the 'bar' variable by one (same as bar = bar - 1;)
```

These operators can also be combined with other operators inside an expression. In this case, their priority is the higher one. Once evaluated, these operators result to the value of their operand before or after being incremented or decremented. This depends on which side of the operand the operator is located.

Example:

```
X1.0 = X2.0++;           // Increment X2.0 by one. The result of this
                        // operation is X2.0 before being
                        // incremented. So, X1.0 is assigned with the value
                        // of X2.0 before being incremented.
X1.0 = ++X2.0;           // Increment X2.0 by one. The result of this operation
                        // is X2.0 after being incremented. So, X1.0 is assigned
                        // with the value of X2.0 after being
                        // incremented.
foo = bar--;             // Decrement 'bar' by one. The result of this operation
                        // is 'bar' before being decremented. So, 'foo' is
                        // assigned with the value of 'bar' before being
                        // decremented.
foo = 23 * (--bar) + 1;  // Complex expression
```

7.3.7 Expressions

An expression is a simple or complex operation involving variables, registers, operators and function calls.

Example:

```
mvt_step = X1.0 + (M6.0 * scale_factor + 1) / get_step_count(120.0F);
```

In this example, the expression combines two variables ('mvt_step' and 'scale_factor'), two registers ('X1.0' and 'M6.0'), two immediate values ('1' and '120.0F'), a function call ('get_step_count') and several operators (=, +, * and /). The usage of brackets helps to group operations and define their priority, as we usually do in standard mathematics expressions.

An expression can be just an assignment of registers or variables:

```
X1.0 = 12;
foo = 23D;
```

Or a simple function call:

```
do_step();           // do_step is a function having no argument
```

In the example above, all expressions end by a semi-colon (;) because they are expression statements (refer to §7.3.8). Expressions do not always end with a semi-colon. When they are used as an argument in a function call or in statements, for example, they do not have any semi-colon (;).

Example:

```
for (i=0; i<10; i++){           // i++ is an expression
    DOUT.0 = i & 0x0001;
    yield();
}
```

When several operators appear in an expression, their priority will determine the order on which they are applied.

Example:

```
X1 = X2 + X3 * X4;
```

There are two operators, '+' and '*'. The '*' operator has a higher priority than the '+' operator, so it will be executed first. This expression is so identical to:

```
X1 = X2 + (X3 * X4);
```

Once again, the execution order can be modified thanks to the brackets.

Example:

```
X1 = (X2 + X3) * X4; // the sum is executed first, its result is multiplied by X4
```

7.3.7.1 Boolean expressions

Boolean expressions are expressions whose result can be true or false.

Example:

```
X1.0 < 12
```

This expression, once evaluated, doesn't result in a value. The result is true if X1.0 is less than 12, or false otherwise.

A boolean expression can combine several tests like this:

```
(X1.0 < 12) && (i == 100) || ((X2.0 + 200) > X1)
```

Boolean operators are allowed only in boolean expressions. Boolean expressions are used in 'if', 'while', 'do' and 'for' statements (refer to [§7.3.8](#) for more information).

7.3.7.2 Commands

All the controller commands described in the previous chapters can be executed in a sequence.

Example:

```
PWR.0 = 1  
IND.0  
MVE.0 = 12000L + 2 * X1.0  
WTM.0  
WBS.0 = DIN, 1 << X2.0
```

From the language point of view, these commands are expressions.

7.3.8 Statements

Statements are used to describe the operations to perform. They are also giving a structure to the code and they are used to control the execution of the sequence. Statements are located in the function body, just after the definition of the local variables (refer to [§7.3.5](#)). Most of the statements available in the sequence language are very similar to the ones defined by the C language.

7.3.8.1 Expression statements

Basically an expression statement is just an expression followed by a semi-colon (;). Most of the time, the expression contains an assignment operator or a function call.

Example:

```

X1 = 23;
actual_position = M6.0 * scale_factor(X2);
start_movement(X3);
MVE.0 = 12000L;
PWR.0=1;

```

Refer to [§7.3.7](#) for more information about the expressions.

7.3.8.2 The block statement

A block statement allows the user to group any number of statements into one statement. All statements enclosed within a single set of braces ('{', '}') are treated as a single statement. You can use a block wherever a single statement is allowed.

Syntax:

```

{
    <statement 1>
    <statement 2>
    <statement 3>
    ...
}

```

In the C language, it is possible to define local variables inside a block statement. This is not allowed in a ETEL sequence.

7.3.8.3 The 'if' statement

The 'if' statement allows the user to execute or not a statement depending on the value of a boolean expression.

Syntax:

```

if (<boolean expression>)
    <statement 1>

```

or

```

if (<boolean expression>)
    <statement 1>
else
    <statement 2>

```

The <statement 1> is executed only if the <boolean expression> is true, otherwise <statement 2> is executed. When more than one statement must be executed, <statement 1> and <statement 2> must be block statements.

Example:

```

if (X1.0 == 12) {
    // all the following statements are executed if X1.0 is 12
    <statement 1a>
    <statement 1b>
    <statement 1c>
    ...
}

```

The statement located after 'else' can be an 'if' statement. This let you manage multiple choices.

Example:

```
if (X1.0 < 10)
    MVE.0 = 12000L;
else if (X1.0 > 10)
    MVE.0 = -12000L;
else                                     // implicitly, X1.0 value is 10
    BRK.0;
```

7.3.8.4 The 'switch' statement

The 'switch' statement allows the user to select which statement must be executed depending on the value of an expression. It works in collaboration with the 'case' and 'break' statements.

Syntax:

```
switch (<expression>) {
    case <value 1>:
        <statement 1a>
        <statement 1b>
        ...
        break;
    case <value 2>:
        <statement 2a>
        <statement 2b>
        ...
        break;
    case ...
    ...
    default:
        <statement 3a>
        <statement 3b>
        ...
        break;
}
```

<statement 1a>, <statement 1b> and the following ones (...) are executed only if the <expression> is equal to <value 1>.

<statement 2a>, <statement 2b> and the following ones (...) are executed only if the <expression> is equal to <value 2>.

It is possible to add as many 'case' statements as needed. Each 'case' statement must have a unique value.

If the <expression> does not correspond to any value defined by the 'case' statements, <statement 3a>, <statement 3b> and the following ones (...) present under 'default', are executed.

The use of 'break' statements is not mandatory. If a 'break' statement is not present, all the following statements will be executed by ignoring the case.

Example:

```
switch (X1.0) {
    case 1:
        <statement 1a>
        <statement 1b>
        // no break here
    case 2:
        <statement 2a>
}
```

In this example, if X1.0 is equal to 1, <statement 1a>, <statement 1b> and <statement 2a> will be executed. If X1.0 is equal to 2, only <statement 2a> will be executed.

7.3.8.5 The 'while' statement

A 'while' statement repeatedly executes a given statement until the conditional expression is false.

Syntax:

```
while (<boolean expression>)  
    <statement>
```

<statement> is executed several times while <boolean expression> is true. <boolean expression> is evaluated before each execution of <statement> to determine whether or not the execution of <statement> must be done. If <boolean expression> is false since the first evaluation, <statement> is never executed.

When more than one statement must be repeatedly executed, <statement> must be a block statement.

Example:

```
while (i < 10) {  
    // execute the following instructions until i >= 10  
    MVE.0 = 10000L * i;  
    WTM.0;  
    i++;           // increment i  
}
```

A 'break' or 'return' statement causes a 'while' statement to end, even when the <boolean expression> is true.

7.3.8.6 The 'do' statement

A 'do' statement repeatedly executes a statement until the conditional expression is false. Because of the order of processing, the statement is executed at least once.

Syntax:

```
do  
    <statement>  
while (<boolean expression>);
```

<statement> is executed before <boolean expression> is evaluated. Further processing of the 'do' statement depends on the value of <boolean expression>. If <boolean expression> is true, the statement is executed again. When <boolean expression> is false, the statement ends.

When more than one statement must be repeatedly executed, <statement> must be a block statement.

A 'break' or 'return' statement causes a 'do' statement to end, even when the <boolean expression> is true.

7.3.8.7 The 'for' statement

A 'for' statement repeatedly executes a statement, as the 'while' and 'do' statements do. The control of the execution depends on three expressions:

- An expression is evaluated before the first iteration of the statement
- A boolean expression determines whether or not the statement should be processed. The statement is repeatedly executed if this boolean expression is true
- An expression is evaluated after each iteration of the statement (often used to increment an iteration counter)

Syntax:

```
for (<expression 1>; <boolean expression>; <expression 2>)  
    <statement>
```

<expression 1> is evaluated only once, before <statement> is executed for the first time. You can use this expression to initialize a variable (typically the iteration counter). If you do not want to evaluate an expression

prior to the first iteration of the statement, this expression can be omitted.

<boolean expression> is evaluated before each iteration of the statement. If it is false, the statement is not processed and control moves to the next statement following the 'for' statement. If <boolean expression> is true, the statement is processed. If <boolean expression> is omitted, it is like if an always true expression was present, and the 'for' statement is not terminated by failure of this condition.

<expression 2> is evaluated after each iteration of the statement. This expression is often used for incrementing, decrementing, or assigning to a variable. This expression is optional.

When more than one statement must be repeatedly executed, <statement> must be a block statement.

A 'break' or 'return' statement causes a 'for' statement to end, even when the <boolean expression> is true. If the <boolean expression> is omitted, a 'break' or 'return' statement must be used to end the 'for' statement.

Example:

```
int do_movement_steps(int number_of_steps)
{
    int i;
    for (i=0; i<number_of_steps; i++) {
        // Do the following movements number_of_steps times.
        // On the first iteration, i = 0, On the second, i = 1, etc.
        // On the last iteration, i = number_of_steps - 1.
        RMVE.0 += 4600L;
        WTM.0;
        // If the DIN is set, stop to make steps (exit from the loop).
        if ((DIN & 0x02) != 0) {
            i++;
            break;
        }
    }
    return i; // Return the real number of step done.
}
```

7.3.8.8 The 'break' statement

A 'break' statement ends a 'switch', 'do', 'for' or 'while' statement and exits from it at any point other than the logical and. A 'break' may only appear on one of these statements.

Syntax:

break;

In a 'switch' statement, the break passes control out of the switch body to the next statement outside the 'switch' statement (refer to [§7.3.8.4](#)).

In a 'while', 'do' or 'for' statement, the 'break' statement ends the loop and moves control to the next statement outside the loop.

Example:

```
for (;;) {
    // no end condition, iterate until a break is
    // reached
    if ((DIN & 0x01) != 0) // test the DIN value
        break; // exit the loop
    yield(); // wait next controller cycle
}
```

In this example, we stay in the loop until the DIN has been set. This is similar to waiting the DIN to be set.

7.3.8.9 The 'continue' statement

A 'continue' statement ends the current iteration of a 'while', 'do' or 'for' statement. Program control is passed from the 'continue' statement to the end of the body loop. A 'continue' statement can only appear within the body of their statements.

Syntax:

```
continue;
```

In a 'for' statement, for example, when the 'continue' statement is reached, the control moves to the third expression in the condition part of the statement (the part enclosed in the brackets), then to the second expression (boolean expression) in the condition part of the statement.

7.3.8.10 The 'return' statement

A 'return' statement ends the processing of the current function and returns control to the caller of the function.

Syntax:

```
return;
```

or

```
return <expression>;
```

A function with a non-void return type should always include a 'return' statement, containing an expression.

For a function of return type 'void', a 'return' statement is not strictly necessary. If the end of such a function is reached without encountering a 'return' statement, control is passed to the caller as if a 'return' statement without an expression were encountered. In other words, an implicit return takes place on completion of the final statement, and controls automatically returns to the calling function. If a 'return' statement is used, it must not contain an expression.

7.3.9 Unit conversion

To convert an ISO value to an increment there are special conversion functions. For each data type there is a specific conversion function

| Increment value type returned by the function | Conversion function |
|--------------------------------------------------|---------------------|
| int | iconv |
| long | lconv |
| float | fconv |
| double | dconv |

The conversion functions accept either two or three arguments:

```
int iconv(isoValue, _unit)           int iconv(isoValue, _unit, axisNumber)
long lconv(isoValue, _unit)          long lconv(isoValue, _unit, axisNumber)
float fconv(isoValue, _unit)         float fconv(isoValue, _unit, axisNumber)
double dconv(isoValue, _unit)        double dconv(isoValue, _unit, axisNumber)
```

If no axis number is given, the compiler does the conversion for the axis on which the sequence is running.

Example:

```
X1.0 = iconv(0.02, _upi);           // convert 0.02 meters into user position
                                     // increments and put the value in X1.0
```

These conversion functions work only with immediate ISO values (refer to §7.2.6) on the first argument. It is not possible to make conversions of value stored in variables or registers, or resulting from expressions:

```
X1.0 = iconv(X2.0, _upi);      // this usage of iconv() is NOT ALLOWED!
X1.0 = iconv(2 * bar, _upi);   // this usage of iconv() is NOT ALLOWED!
X1.0 = iconv(0.2F, _upi, 3);   // this usage of iconv() is NOT ALLOWED!
                                // the first argument must be an iso value
```

Here is the meaning of the main units:

| Units | Meaning |
|-------|------------------------------------------------------|
| cur | Current |
| cur2 | Current * Current |
| mlti | Number of MLTI in the given time (e.g. use for WTT) |
| msec | Conversion for millisecond |
| temp | Conversion for temperature |
| uai | User Acceleration Increment |
| ufai | User Friendly Acceleration Increment (Interpolation) |

| Units | Meaning |
|-------|--------------------------------------------------|
| ufpi | User Friendly Position Increment (Interpolation) |
| ufsi | User Friendly Speed Increment (Interpolation) |
| ufti | User Friendly Time Increment (Interpolation) |
| upi | User Position Increment |
| upi2 | User Position Increment for secondary encoder |
| usi | User Speed Increment |
| volt | Voltage |

7.3.10 Predefined functions

Functions (refer to §7.3.5) can be defined by the developer and can be called from the sequence itself. ETEL's sequences have also a set of predefined function. This is similar to the standard library available when programming in C, where a set of functions is available to the C developer. In C, the developer has to insert a #include directive to have access to the standard library. On the ETEL sequences, predefined functions are available without doing anything.

Here is the list of predefined functions with their prototype (arguments and return types) and their behavior. The names of these functions have been chosen as close as possible to the ones of the standard C library.

7.3.10.1 start

This function starts a thread. On UltimET, there are 3 threads available: the number 1, 2 and 3. The thread_number argument defines which one of these 3 threads is started. When started, the thread executes (call) the function given as first argument (refer to §7.4.4 for more information).

Prototype:

```
void start(function f, int thread_number);
```

7.3.10.2 exit

This function stops the thread that calls it. Calling this function has the same behavior of executing the END command.

Prototype:

```
void exit(void);
```

7.3.10.3 yield

When this function is called, the execution of the calling thread is suspended until the next controller cycle.

Prototype:

```
void yield(void);
```

Example:

```
// In the following code, X1.0 is incremented by 1 on each controller cycle.
// In other words, X1.0 is counting the number of cycles.
```

```
for (;;) {
    X1.0++;
    yield();
}
```

7.3.10.4 *abs, labs, fabs and fabsf*

These functions compute and return the absolute value of the value given as argument. Each function has a specific value type.

Prototype:

```
int abs(int value);
long labs(long val);
double fabs(double val);
float fabsf(float val);
```

7.3.10.5 *sqrt and sqrtf*

The sqrt function returns the square root value of the value given as argument.

Prototype:

```
double sqrt(double val);
float sqrtf(float val);
```

7.3.10.6 *Trigonometric functions*

These functions compute the basic trigonometric operations. The angles must be expressed in radians.

Prototype:

| | |
|------------------------------------------|----------------------------------------|
| double sin(double val); | float sinf(float val); |
| double cos(double val); | float cosf(float val); |
| double tan(double val); | float tanf(float val); |
| double asin(double val); | float asinf(float val); |
| double acos(double val); | float acosf(float val); |
| double atan(double val); | float atanf(float val); |
| double atan2(double y, double x); | float atan2f(float y, float x); |

7.3.10.7 *rand*

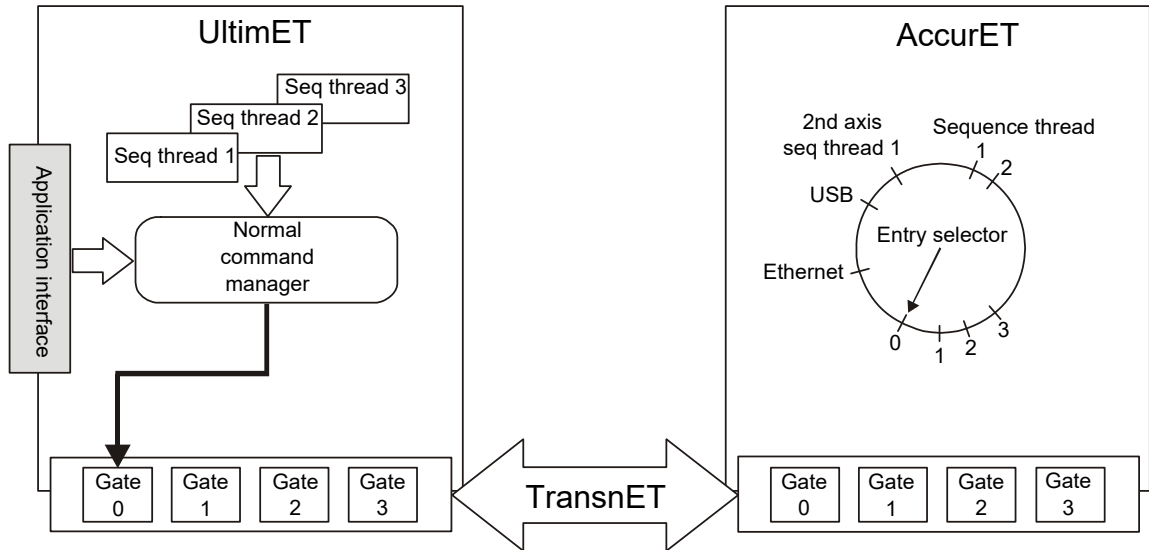
The rand function returns a pseudo-random integer value in the range $[0, 2^{31} - 1]$.

Prototype:

```
int rand(void);
```

7.3.11 Sequence gates

In the structure code, it is possible to allow each sequence thread to send commands through their 'own' gate. A gate can be considered as a separate channel that starts from the UltimET normal commands manager and finishes in a specific AccurET entry. Each gate has its own wait mechanism and does not block other gates when waiting. there are 4 gates in the UltimET. Gate 0 is the default gate for all normal commands, for the application interface as well as for each sequence thread.



Two commands are used to manage the gates:

***gga; // get gate (gga)** can be executed by each sequence thread. This command assigns a gate to a thread: thread 1 uses gate 1, ... The commands sent by the PC application keep using gate 0.

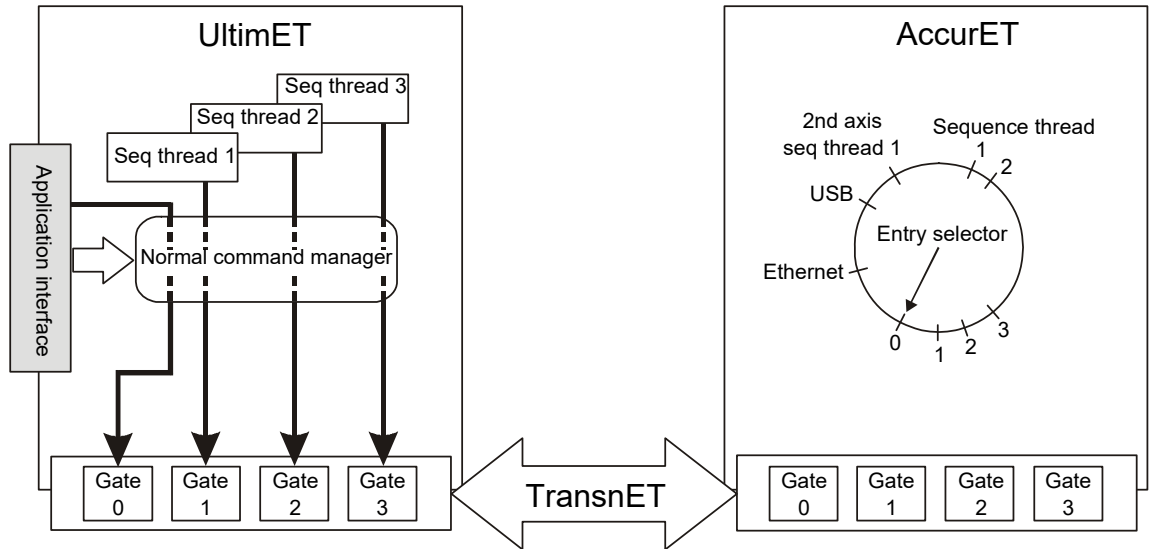
***fga; // free gate (fga)** disables (return to gate 0) this feature.

The monitoring ***M516** allows the user to know the gate number associated to each thread.

| M | Name | Comment |
|-------|-------------------------------------------|----------------------------------------------------------------------------------------------------|
| *M516 | gate number associated to a thread number | Gives the gate number associated to each thread (depth 0 = application bus, depth 1 = thread 1...) |

Example 1:

In this example, each UltimET sequence thread has executed *GGA command.



Example 2:

The first sequence manages the movements by sending MVE and WTM commands to the gate 1.
 The second sequence monitors the i2t value and can immediately reduce the speed the CAM on gate 2 without taking care of the WTM commands.
 The third sequence monitors security parameters (DIN) and stops the motors if needed through the gate 3.

7.3.12 Inputs / outputs management

7.3.12.1 Local inputs / outputs

- **Digital inputs**

In addition to the functionality already existing (DIN), new functions have been introduced to facilitate and standardize the access to these local digital inputs. These functions use the same data as the DIN register alias and follow the same rules concerning the reserved inputs by other features (like protection for example).

These inputs are also accessible through the following functions:

```
int GETDINS(io_idx)
int GETMDINS(io_firstidx, io_number)
```

io_idx: index of the input starting from 0.

io_firstidx: index of the first input in the mask.

io_number: number of inputs in the mask.

- **Digital outputs**

In addition to the functionality already existing (DOUT), new functions have been introduced to facilitate and standardize the access to these local digital outputs. These functions use the same data as the DOUT register alias and follow to the same rules concerning the reserved outputs by other features (like protection for example)

These outputs are also accessible through following functions:

```
int GETDOUT(io_idx)
int GETMDOUT(io_firstidx, io_number)
void SETDOUT(io_idx)
void RSTDOUT(io_idx)
void SETMDOUT(io_firstidx, io_number, value)
```

io_idx: index of the output starting from 0.

io_firstidx: index of the first output in the mask.

io_number: number of outputs in the mask.

value: mask to apply on the digital outputs

7.3.12.2 External inputs / outputs

- **Enabling the update of the I/Os**

The *EIOSTA command and the monitoring *M764 (or *ISEIOSTA alias) are used in the compiled sequences like in the terminal (refer to [§4.11.2.2](#)).

- **Digital inputs**

The following functions allow a direct access to a specific input:

| | |
|-----------------------------------|-----------------------------------------------|
| Reading a digital input: | int GETEDINS(int io_idx) |
| Reading a mask of digital inputs: | int GETMEDINS(int io_firstidx, int io_number) |

io_idx: index of the input starting from 0.

io_firstidx: index of the first input in the mask.

io_number: number of inputs in the mask.

- **Digital outputs**

A direct access to a specific output is done with the following functions:

| | |
|------------------------------------|---------------------------|
| Set one external digital output: | void SETEDOUT(int io_idx) |
| Reset one external digital output: | void RSTEDOUT(int io_idx) |

| | |
|-------------------------------------------|-----------------------------------------------------------|
| Read one external digital output: | int GETEDOUT(int io_idx) |
| Apply a mask on external digital outputs: | void SETMEDOUT(int io_firstidx, int io_number, int value) |
| Get mask external digital outputs: | int GETMEDOUT(int io_firstidx, int io_number) |
| Get actual state of one digital output: | int GETEDOUTS(int io_idx) |
| Get mask external digital outputs state: | int GETMEDOUTS(int io_firstidx, int io_number) |

io_idx: index of the output starting from 0.

io_firstidx: index of the first output in the mask.

io_number: number of outputs in the mask.

value: mask to apply on the digital outputs.

• Analog inputs

These inputs are also accessible with the following functions in compiled sequences:

| | |
|-------------------------------------------------------|-----------------------------|
| Read the raw value of an external analog input: | int GETEAINRS(int io_idx) |
| Read the converted value of an external analog input: | float GETEAINCS(int io_idx) |

io_idx: index of the inputs starting from 0

• Analog outputs

These outputs can also be set with the following functions in compiled sequences (io_idx starting from 0):

| | |
|----------------------------------------------------------------------------|-----------------------------------------|
| Set the raw value of an external analog output: | void SETEAOUTR(int io_idx, int value) |
| Set the converted value of an external analog output: | void SETEAOUTC(int io_idx, float value) |
| Read the raw value of an external analog output: | int GETEAOUTR(int io_idx) |
| Read the converted value of an external analog output: | float GETEAOUTC(int io_idx) |
| Read the actual state of the raw value of an external analog output: | int GETEAOUTRS(int io_idx) |
| Read the actual state of the converted value of an external analog output: | float GETEAOUTCS(int io_idx) |

io_idx: index of the output starting from 0.

value: either the raw or converted value of the analog outputs

• Access to WAGO registers

There is a possibility to access all the addresses of the MODBUS interface of the WAGO module. These functionalities allow the user to have access to the registers configuration of the module or to exchange data with a PLC program which may be running on the WAGO controller. To do so, the **SETEREG** and **GETEREG** functions must be used.

These functions are available in the compiled sequences through:

```
void SETEREG(register_address, value)
Int GETEREG(register_address, word_count, index_of_desired_word)
```

value: value to set in the WAGO register

register_address: address of the WAGO register

word_count: number of words to read

index_of_desired_word: index of the desired word

The use of GETEREG is a little bit special because the access to some WAGO registers is not so trivial. For example registers 1029 has 9 data at the same address. It is not possible to have access directly to the third data. To access this third data, you must read the 9 data and keep only the third one: GETEREG(0x1029,9,2).

• Setting a watchdog

The external I/O modules often have a watchdog that can be programmed to disable (in case of WAGO, set to 0) the outputs if the controller/coupler does not receive any messages for a given time. On WAGO couplers, this requires setting a couple of registers, which can be achieved using the UltimET command *SETEREG(register_address, value).

Remark: This feature has been successfully tested on a coupler type 750-352 and 750-362. For other type types of couplers please check with Wago.

Example:

Below is an ETEL sequence to program and enable a controller/coupler watchdog:

```

SETEREG(4106,0)           // disable any active watchdog (Register Address 0x100A)
SETEREG(4104,43605)       // stop the watchdog (sic) (Register Address 0x1008)
*WTT=1.0                  // wait for previous command to be taken into account
                           // (depends on the WAGO product, subject to change with
                           // the type of WAGO coupler)
SETEREG(4096,50)          // set watchdog time-out, to 5 seconds in this case (time-
                           // out is specified in 100 s of milliseconds on WAGO)
                           // (Register Address 0x1000)
SETEREG(4106,1)           // enable the watchdog(Register Address 0x100A)

```

And the sequence to deactivate a watchdog could be:

```

SETEREG(4106,0)           //disable the watchdog(Register Address 0x100A)
SETEREG(4104,43605)       //stop the watchdog (Register Address 0x1008)

```

7.4 How to start a sequence

There are several ways to start a sequence thread. The basic way is to send a JMP command to the AccurET or UltimET controller. This command can be sent from ComET, from an application running on the PC or from a sequence running on the UltimET Light. It is also possible to start a sequence thread automatically when the AccurET or UltimET controller is switched on. Finally, a sequence thread can be started from another thread. All these method are described in the next paragraphs.

It is not allowed to start a sequence thread that is already running. To know how to stop a running thread, refer to [§7.5](#).

7.4.1 Start a sequence thread from ComET

You can start a sequence on a controller by sending a JMP command. This can be done from the ComET's Terminal. The syntax of the **JMP** command is as following:

| Command | <P1> | <P2> | Comment |
|----------------------------|-----------------|---------------|---------------------------------------------------|
| *JMP[.<axis>] = <P1>,<P2>] | Function number | Thread number | Start function number <P1> in thread number <P2>. |

Remark: If <P2> is not mentioned or equal to zero, the jump will be executed by the current thread (the one which is executing the instruction). If the command is sent by the PCI/PCIe or TCP/IP bus, the default thread is the thread number 1.

Example:

To start a sequence thread on the axis 5, the following command must be sent:

```
JMP.5 = 25
```

When the axis 5 receives this command, it starts the execution of the function named 'func25' in the first sequence thread (thread number 1).

The second parameter of the command indicates which thread has to be started. There are three threads available on the UltimET Light controllers, so, the value of this parameter can be 1, 2 or 3. If this parameter is omitted or set to zero, the thread 1 is started.

Example:

```
*JMP = 20, 2
```

This command starts the execution of the function named 'func20' in the thread 2 of the UltimET Light.

7.4.2 Start a sequence thread from an application running on the PC

Applications running on the PC are normally communicating with AccurET or UltimET controllers through the EDI libraries. Thanks to these library, it is possible to start a sequence thread on any controller. To do so, the application has to call `dsa_execute_sequence_s()` or `dsa_execute_sequence_a()`. Please refer to the '**EDI User Manual**' for more information.

7.4.3 Start a sequence on the AccurET from a sequence present in the UltimET

The JMP command can be used on sequences as well as on the ComET's Terminal. A sequence in the UltimET can then start a sequence thread on an AccurET by executing the command `JMP.<axis> = <function>, <thread>`.

7.4.4 Start a sequence thread from another thread

To take advantage of the multi-thread architecture, a sequence thread must be able to start another sequence thread. This can be done with the function 'start()'.

The 'start()' function has two arguments: the function executed by the new thread and the new thread number.

Example:

```
void supervisor(void)
{
    for (;;) {
        if (M6.0 > X1.0)
            DOUT.0 = 0x10;
    }
}

void func20(void)
{
    X1.0 = 12000;
    start(supervisor, 2); // start the thread number 2 to execute supervisor()
    MVE.0 = 0L;
    WTM.0;
    MVE.0 = 25000L;
    WTM.0;
    END.0 = 2;           // stop the thread number 2
}
```

In this example, the function named 'func20' starts a new thread (the thread number 2) that executes the function named 'supervisor'. Both functions run then in parallel (at the same time).

7.4.5 Start a sequence thread automatically

When the controller is switched on, it checks for the presence of a function named "autostart". If it is found, the thread number 1 is automatically started to execute the autostart function.

The 'autostart' function is a function called from outside the sequence itself, so it must have the following prototype: **void autostart(void)**.

7.5 How to stop a sequence thread

The **END** command allows the user to definitively stop the execution of a sequence's thread. The ***END**

command can be sent from ComET, from the host PC application or from another sequence.

| Command | Comment |
|----------------------|------------------------------------------------------------------------------------------|
| *END[.<axis>]=[<P1>] | Stops the sequence's thread defined by <P1> or stops all threads if command without <P1> |

Remark: This feature represents a compatibility break if the command *END.<axis> was used without parameter with version $\leq 2.07B$.

*HLB, *HLO and *HLT commands stop the execution of the sequence (refer to §6.4.5.4). By default, in this case, all threads are stopped by a single command. The execution can also be stopped in case of error (refer to §7.6). However, with the parameter **K144**, it is possible for a thread to be not stopped by the *HLB, *HLO and *HLT commands (normal or urgent).

| K | Bit# | Value | Comment |
|-------|------|-------|-----------------------------------------------------------------------------------------|
| *K144 | 1 | 2 | Mask for thread 1: if set to 1, the thread 1 is not stopped by HLT, HLB or HLO command. |
| | 2 | 4 | Mask for thread 2: if set to 1, the thread 2 is not stopped by HLT, HLB or HLO command. |
| | 3 | 8 | Mask for thread 3: if set to 1, the thread 3 is not stopped by HLT, HLB or HLO command. |

7.6 Error management

When an error occurs on the UltimET controller, each active thread can independently react in a way defined by the user. The possible reactions are:

- Stop the thread.
- Jump to an error routine.
- Ignore the error and just continue the execution.

Each thread can have its own error routine and its own reaction to the error. This is configured thanks to the parameter **K142**.

Syntax:

*K142:thread.<axis>=<label_error>.

| K | <label_error> | Comment |
|-------|------------------|-----------------------------------------------------------------------------------------------------------------------------|
| *K142 | <routine_number> | The thread is stopped and the error routine defined by <routine_number> is executed. |
| | -1 | Thread 1 jumps to the routine called errhandler if it exists, otherwise the thread is stopped. Threads 2 and 3 are stopped. |
| | -2 | The thread is not stopped in case of error. The execution just continues and no error routine is executed. |

<routine number> is a number from 0 to 1023. It defines the error routine as the function named 'func<routine_number>'. Each depth of *K142 represents a thread:

| | |
|---------|-----------------------------------------------------------------------------------------------|
| *K142:0 | Not used |
| *K142:1 | Definition of the error routine of the thread 1. Default value for depth1 is 80 (*K142:1=80). |
| *K142:2 | Definition of the error routine of the thread 2. Default value for depth 2 is -1. |
| *K142:3 | Definition of the error routine of the thread 3. Default value for depth 3 is -1. |

Example:

```
void func120(void)                // error routine
{
    DOUT.0 = 0;
    X0.0 = M7.0;
}

void func20(void)
{
    K142:1.0 = 120;                // The error routine if the function 'func120'
                                   // So, if an error occurs, func120() is called.
    PWR.0 = 1;
    IND.0;
    MVE.0 = 12000L;
    ...
}
```

}

When an error is raised, no threads are created, even if an error routine is defined in K142. There is no spontaneous start of a thread.

A thread jumps to the error routine only on the first error. If additional errors are raised when the controller is already in error state, the error routine is neither interrupted nor restarted. Except if the error routine clears all the errors. In this case, the error routine is restarted if a new error is raised.

7.6.1 Predefined error routine

For applications running only on **thread 1**, the user can define an error routine named "errhandler". K142:1 must be set to -1 to enable it. Caution: the "errhandler" routine is only available for the first thread and must not be used with several threads.

Example:

```
void errhandler(void) // error routine
{
    // the following code is executed when an error occurs
    ...
}
```

7.6.2 ERR command

From firmware 2.07A, it is possible to add an optional parameter to the command **ERR.<axis>[=<P1>]**. It allows the user to set application errors through the sequence for example. In that case, <P1> **must be a negative value**.

- If <P1> is a negative value, M64.<axis>=<P1>, the error message in M95.<axis> is 'EXTERNAL ERROR' and the message on ComET application is 'ERROR <P1>: unknown'.
- If the command is without parameter <P1>, the error message in M95 is 'EXTERNAL ERROR' and the message on ComET application is: 'ERROR 116': This error is generated by the ERR command'.

| Command format | Comment |
|----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ERR.<axis> [= <P1>] | Puts the controller in error mode (M64=116). If <PAR1> is a negative value, the error code in M64.<axis> is not 116 but the value of <PAR1> defined by the user. |

7.7 Performance aspects

There are two factors, apart from the actual sequence code itself, that have an influence on how fast the sequence is executed:

- The time given to the compiled sequence scheduler by the firmware
- The kind of processor memory used for the execution stack

7.7.1 Allocation of time to the compiled sequence

7.7.1.1 Default configuration

The default strategy is to ensure that the execution time taken by the high priority task, including running the compiled sequence, leaves enough time for lower priority tasks to run within a TransnET cycle. This depends on how much interpolation time is required on the UltimET. The time allotted to the compiled sequence in a given cycle is calculated such that:

- If no interpolation is done in the UltimET, the high priority tasks takes no more than 50% of the cycle time (leaving the other 50% for low priority or background tasks),
- If one interpolation group is running, the high priority tasks takes no more than 2/3rd of the cycle time (leaving the other third for low priority or background tasks),
- If two interpolation groups are running, the high priority tasks takes no more than 80% of the cycle time (leaving 20% for low priority or background tasks).

Consequently, the effective time a sequence has to run depends on what else is being executed by the firmware.

7.7.1.2 User configuration

It is possible for the user to optimize the given amount of time for the execution of the compiled sequence by setting the parameter ***K200**. In this way, the same execution time for each high priority cycle is guaranteed. It is therefore more adapted for real-time application than the default configuration described above.

| K | Name | Comment |
|--------------|------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *K200 | Time allowed for compiled sequence | The value is a multiple of 3.75 ns. The typical value is set to 8000, which corresponds to 30 µs. The minimum and maximum values are 0 and 13333, i.e. 0 µs and 50 µs. |

The compiled sequence scheduler splits this time into the number of threads running in the sequence. It is possible to monitor the allotted time for the threads and the effective used time thanks to the monitorings **M201** and **M202**.

| M | Name | Comment |
|--------------|-------------------------------|---------------------------------------------------------------------------------------------------------|
| *M201 | Sequence thread allotted time | Time quota for the threads. The <depth> corresponds to the thread number (1 or 2 – 0 not used). |
| *M202 | Time used by the thread | Effective time used by the threads. The <depth> corresponds to the thread number (1 or 2 – 0 not used). |

It is the user's responsibility to set the compiled sequence time slot so that the high priority task do not overrun the cycle time. Should this happen, the error ***M64=4120** (high priority interrupt overflow) is raised.

7.7.2 Compiled sequence stack management

7.7.2.1 Setting

To increase the performance of the compiled sequences, it is possible to place the compiled sequence thread stacks in internal memory. This yields approximately a 2.5 fold increase in execution speed, depending on the content of the sequence. When in internal RAM, the stacks size is however significantly reduced. It contains 256 entries (32-bit integers), whereas when in external RAM it is 2048 slots. This means that sequences running in the fast configuration cannot have as many call depths as when running with the stacks in external RAM. The stack location is selected by setting the parameter ***K415**.

| K | Name | Value | Comment |
|-------------|------------------------------------------|-------|--------------------------------------------------------------------------------------|
| K415 | Enable fast stack for compiled sequences | 0 | Sequence stack in external memory (default case). |
| | | 1 | Sequence stack in internal memory. Warning: the stack size is significantly reduced. |

Warning: This feature is only available on the PCI and PCIe versions of UltimET Light. Setting ***K415** on an UltimET TCPIP raises the error ***M64=3006** (Command not available).

A modification of ***K415** becomes active after a save operation (SAV.<axis>=2) and a reboot of the controller. The monitorings ***M417** and ***M418** allow the user to monitor the compiled sequence threads stack sizes and usage.

| M | Name | Comment |
|--------------|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *M417 | Total stack size | The <depth> corresponds to the thread number 1 or 2 (0 is not used). This register is equal to 256 when stacks in internal RAM and 2048 when stacks in external RAM. |
| *M418 | Available free stack space | The <depth> corresponds to the thread number 1 or 2 (0 is not used). |

7.7.2.2 Errors and restrictions

If a stack overflows, the error ***M64=410** (sequence error: stack overflow) is raised. This error cannot be reset by a ***RST** command and the board must be rebooted. It is due to too many nested calls or/and local variables. In that case, it is necessary to rework the sequence.

Chapter F: Appendixes

8. Commands reference list

(1): once sent, this command is stored in the interpolation commands buffer. The others are directly executed.

Remark: The [,<axis>] field is optional.

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|-------------------------------------------|---------|--------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|-------------------------------|
| *ASR[,<AXIS>] = <P1>, <P2>, <P3> | All | 231 | | Gives slot (offset in the TransnET cyclic data) where the controller can read cyclic data RTV. <P1> is the Mx450 to Mx465, where we want to put the read data, <P2> is the slot offset from where it reads the low and <P3> the high part in case of 64bits | | |
| | | | 1 | Defines the monitoring register that is read as a RTV register in the TransnET cyclic data: M450 to M465, or MF450 to MF465, or ML450 to ML457 or MD450 to MD457 | | |
| | | | 2 | Defines the offset slot address in the TransnET cyclic data for the read RTV register, lower part | | |
| | | | 3 | Defines the offset slot address in the TransnET cyclic data for the read RTV register, higher part. Optional, only available for ML or MD | | |
| *ASW[,<AXIS>] = <P1>, <P2>[, <P3>] | All | 232 | | Sets slot (offset in the TransnET cyclic data) where the controller writes cyclic data RTV | | |
| | | | 1 | Defines the register to be written by the controller (RTV) in the TransnET cyclic data. If the value=0, the register is not written anymore in the TransnET cyclic data | | |
| | | | 2 | Defines the offset in the TransnET DPRAM where the controller writes the register (RTV) lower part | | |
| | | | 3 | Defines the slot in the TransnET cyclic data where the controller writes the register, MSL part (RTV). Only available for 64bits registers | | |
| *CH_BIT_REG32[,<AXIS>] = <P1>, <P2>, <P3> | All | 90 | | Allows to write any bit (set bits and/or clear bits) <P2> with defined value <P3> of any 32 integer register <P1> without modifying the other bits the register | | |
| | | | 1 | Destination register | | |
| | | | 2 | Mask of the bit that can be modified | | |
| | | | 3 | Mask value of the bit | | |
| *CHKDISTGRT = <P1>, <P2>, <P3> | All | 563 | | Checks the values in the registers specified in the first 2 parameters are separated by the value given in the third. | | |
| | | | 1 | Defines the first register to compare | | |
| | | | 2 | Defines the second register to compare | | |
| | | | 3 | Defines the minimum distance between the two registers | | |
| *CLRWAIT = <P1> | All | 180 | | Clear busy state due to a wait command | | |
| | | | 1 | mask of the entry where the wait could be cleared (Not allowed for the sequence) | Bit 1 | Sequence thread 1 |
| | | | | | Bit 2 | Sequence thread 2 |
| | | | | | Bit 3 | Sequence thread 3 |
| | | | | | Bit 16 | Application bus PCI |
| | | | | | Bit 17 | Application bus TCP/IP Port 1 |
| | | | | | Bit 18 | Application bus TCP/IP Port 2 |
| | | | | | Bit 19 | Application bus TCP/IP Port 3 |
| *CLX[,<AXIS>] = <P1> | All | 17 | | Clears all user variables (X, XF, XL, XD, Y register depending <P1>) | | |

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|------------------------------------------------|---------|-----------|--------------|--------------------------------------------------------------------------------------------------------------------------|-------------------|----------------------------------------------------|
| | | | 1 | Clears registers X and Y | 0 | Clears X, XF, XL, XD and Y registers at all depths |
| | | | | | Bit 0 | Clears X user variables at all depths |
| | | | | | Bit 1 | Clears XF user variables at all depths |
| | | | | | Bit 2 | Clears XL user variables at all depths |
| | | | | | Bit 3 | Clears XD user variables at all depths |
| | | | | | Bit 4 | Clears Y registers at all depths |
| *EIOMAXURST | All | 755 | | Reset the observed maximum time between IO updates (*M706) | | |
| *EIOSTA = <P1> | All | 750 | | Start external IO read/write cycle | | |
| | | | 1 | Enable the external IO read/write cycle | | |
| *EIOWDEN = <P1> | All | 720 | | Enable/disable external IO watchdog | | |
| | | | 1 | Enable/Disable external IO watchdog | | |
| *EIOWDSTP | All | 721 | | Stop the external IO watchdog | | |
| *EIOWDTIME = <P1> | All | 722 | | Set the external IO watchdog timeout | | |
| | | | 1 | The external IO watchdog timeout | | |
| *END[.<AXIS>] = <P1> | All | 0 | | Stops the sequence thread defined by <P1> or stops all threads if command without <P1> | | |
| | | | 1 | Thread to stop | | |
| *ERR = <P1> | All | 80 | | Puts UltimET light in error mode (error number (M64) = param 1 of the command) | | |
| | | | 1 | Error number, which will appear in *M64 | | |
| *FGA | All | 313 | | Release the gate assigned by the GGA command | | |
| *FWD = <P1>, <P2>, <P3> [, <P4>, ..., <P9>] | All | 520 | | Forward any command to an axis mask with all its parameters. This command may be forwarded as an urgent or normal record | | |
| | | | 1 | Command axis mask | | |
| | | | 2 | Command number | | |
| | | | 3 | Is this command an urgent command (0 = false, 1=true) | | |
| | | | 4 | Command parameter 1 | | |
| | | | 5 | Command parameter 2 | | |
| | | | 6 | Command parameter 3 | | |
| | | | 7 | Command parameter 4 | | |
| | | | 8 | Command parameter 5 | | |
| | | | 9 | Command parameter 6 | | |
| GETDINS(<P1>) : <P2> | All | 738 | | Get the actual value of a specified digital local input | | |
| | | | 1 | Index of the digital local input to get | | |
| | | | 2 | Value of the input | | |

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|----------------------------------|---------|-----------|--------------|--------------------------------------------------------------------------|-------------------|-------------------|
| GETDOUT(<P1>) : <P2> | All | 735 | | Get the value of a specified digital local output | | |
| | | | 1 | Index of the digital local output to get | | |
| | | | 2 | Value of the output | | |
| GETEAINCS(<P1>) : <P2> | All | 746 | | External IO: get the actual converted value of a specified analog input | | |
| | | | 1 | Index of the converted analog input to get | | |
| | | | 2 | Value of the converted analog input | | |
| GETEAINRS(<P1>) : <P2> | All | 745 | | External IO: get the actual raw value of a specified analog input | | |
| | | | 1 | Index of the raw analog input to get | | |
| | | | 2 | Value of the raw analog input | | |
| GETEAOUTC(<P1>) : <P2> | All | 741 | | External IO: get the set converted value of a specified analog output | | |
| | | | 1 | Index of the set converted analog output to get | | |
| | | | 2 | Value of the set converted analog output | | |
| GETEAOUTCS(<P1>) : <P2> | All | 743 | | External IO: get the actual converted value of a specified analog output | | |
| | | | 1 | Index of the actual value of the converted analog output to get | | |
| | | | 2 | Value of the actual converted analog output | | |
| GETEAOUTR(<P1>) : <P2> | All | 740 | | External IO: get the set raw value of a specified analog output | | |
| | | | 1 | Index of the set raw analog output to get | | |
| | | | 2 | Value of the set raw analog output | | |
| GETEAOUTRS(<P1>) : <P2> | All | 742 | | External IO: get the actual raw value of a specified analog output | | |
| | | | 1 | Index of the actual value of the raw analog output to get | | |
| | | | 2 | Value of the actual raw analog output | | |
| GETEDINS(<P1>) : <P2> | All | 765 | | External IO: get the actual value of a specified digital input | | |
| | | | 1 | Index of the digital input to get | | |
| | | | 2 | Value of the input | | |
| GETEDOUT(<P1>) : <P2> | All | 760 | | External IO: get the set value of a specified digital output | | |
| | | | 1 | Index of the digital output to get | | |
| | | | 2 | Value of the output | | |
| GETEDOUTS(<P1>) : <P2> | All | 764 | | External IO: get the actual value of a specified digital output | | |
| | | | 1 | Index of the digital output actual value to get | | |
| | | | 2 | Actual value of the output | | |
| GETEREG(<P1>, <P2>, <P3>) : <P4> | All | 726 | | Get a register from the external IO system | | |
| | | | 1 | Address of the register in the WAGO controller | | |
| | | | 2 | Number of registers to read from the WAGO controller at the same time | | |

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|----------------------------------|---------|-----------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-------------------|
| GETMDINS(<P1>, <P2>) : <P3> | All | 734 | 3 | Index of the register to return (from 0) | | |
| | | | 4 | Value of the register | | |
| | | | | Get the state of a set (mask) of local digital inputs | | |
| | | | 1 | First input of the mask | | |
| GETMDOUT(<P1>, <P2>) : <P3> | All | 732 | 2 | Number of input of the mask | | |
| | | | 3 | Value of the masked input | | |
| | | | | Get the state of a set (mask) of local digital outputs | | |
| | | | 1 | First output of the mask | | |
| GETMEDINS(<P1>, <P2>) : <P3> | All | 758 | 2 | Number of output of the mask | | |
| | | | 3 | Value of the masked output | | |
| | | | | External IO: get the actual state of a set (mask) of external digital inputs | | |
| | | | 1 | First input of the mask | | |
| GETMEDOUT(<P1>, <P2>) : <P3> | All | 757 | 2 | Number of input of the mask | | |
| | | | 3 | Value of the masked input | | |
| | | | | External IO: get the set value of a set (mask) of external digital outputs | | |
| | | | 1 | First output of the mask | | |
| GETMEDOUTS(<P1>, <P2>) : <P3> | All | 766 | 2 | Number of output of the mask | | |
| | | | 3 | Actual output states corresponding to the mask | | |
| | | | | External IO: get the actual state of a set (mask) of external digital outputs | | |
| | | | 1 | First output of the mask | | |
| *GGA | All | 312 | | Assign a new gate to the current thread | | |
| *HLB[.<AXIS>] | All | 121 | | Stops the sequences, stops the interpolated movements with a defined deceleration and disables the interpolation mode at the end of the movement brake | | |
| *HLO[.<AXIS>] | All | 119 | | Stops the sequences, stops the interpolated movements with an infinite deceleration and disables the interpolation mode | | |
| *HLT[.<AXIS>] | All | 120 | | Stops the sequences, stops the interpolated movements with an infinite deceleration and disables the interpolation mode | | |
| *IABSCOORDS = <P1>, ..., <P5> | All | 556 | | Assigns the coordinates of the current position of the axes of the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | X axis abs coordinate | | |
| | | | 3 | Y axis abs coordinate | | |
| | | | 4 | Z axis abs coordinate | | |

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|--------------------------|---------|-----------|--------------|-----------------------------------------------------------------------------------------------------------------------|-------------------|-------------------|
| | | | 5 | Theta axis abs coordinate | | |
| *IABSMODE = <P1>, <P2> | All | 555 | | Activates (1) or clears (0) the absolute reference coordinates mode of the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Sets (1) or clears (0) the abs ref mode | | |
| *IBEGIN = <P1> | All | 553 | | Switches the selected interpolation group into interpolated mode | | |
| | | | 1 | Interpolation group | | |
| *IBRK = <P1> | All | 653 | | Stops the selected interpolation group, with the current deceleration value | | |
| | | | 1 | Interpolation group | | |
| *ICCW = <P1>, ..., <P5> | All | 1041 | | Adds a Counter-ClockWise circular move, with center definition to the trajectory for the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | X axis coordinate of circle segment end | | |
| | | | 3 | Y axis coordinate of circle segment end | | |
| | | | 4 | X axis coordinate of circle center | | |
| | | | 5 | Y axis coordinate of circle center | | |
| *ICCWR = <P1>, ..., <P4> | All | 1027 | | Adds a Counter-ClockWise circular move, with radius definition to the trajectory for the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | X axis coordinate of circle segment end | | |
| | | | 3 | Y axis coordinate of circle segment end | | |
| | | | 4 | Circle radius | | |
| *ICLRB = <P1> | All | 657 | | Clears the interpolation command buffer of the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| *ICONC = <P1> | All | 1030 | | Starts the concatenated mode in the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| *ICONT = <P1> | All | 654 | | Restarts the previously stopped interpolation group, with the current acceleration value | | |
| | | | 1 | Interpolation group | | |
| *ICW = <P1>, ..., <P5> | All | 1040 | | Adds a ClockWise circular move, with center definition to the trajectory for the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | X axis coordinate of circle segment end | | |
| | | | 3 | Y axis coordinate of circle segment end | | |
| | | | 4 | X axis coordinate of circle center | | |
| | | | 5 | Y axis coordinate of circle center | | |
| *ICWR = <P1>, ..., <P4> | All | 1026 | | Adds a ClockWise circular move, with radius definition to the trajectory for the selected interpolation group | | |

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|--------------------------|---------|-----------|--------------|---------------------------------------------------------------------------------|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| | | | 1 | Interpolation group | | |
| | | | 2 | X axis coordinate of circle segment end | | |
| | | | 3 | Y axis coordinate of circle segment end | | |
| | | | 4 | Circle radius | | |
| *IEND = <P1> | All | 554 | | Switches the selected interpolation group out of interpolated mode | | |
| | | | 1 | Interpolation group | | |
| *ILINE = <P1>, ..., <P5> | All | 1025 | | Adds a linear move to the trajectory for the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | X axis coordinate of the segment end | | |
| | | | 3 | Y axis coordinate of the segment end | | |
| | | | 4 | Z axis coordinate of the segment end | | |
| | | | 5 | Theta axis coordinate of the segment end | | |
| *ILOCK = <P1> | All | 1044 | | Locks the selected interpolation group for the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| *IMARK = <P1>, ..., <P5> | All | 1039 | | Sends a mark command introduced in the corresponding interpolation group buffer | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Mark number | | |
| | | | 3 | Action | 0 | Does nothing |
| | | | | | 1 | Sets the bits of the user status defined by the parameter <P4> |
| | | | | | 2 | Clears the bits of the user status defined by the parameter <P4> |
| | | | | | 3 | Starts the sequence function defined with parameter <P4>, in the thread defined by parameter <P5>. |
| | | | | | 4 | Sets an integer user variable whose index (bit# 0-23) and depth (bit# 24-31) are given in parameter <P4> with the value of parameter <P5> |
| | | | | | 5 | Sets / resets digital outputs. The state of the outputs is given in <P4> (Bit 0-15: set, bit 16-31: reset). It is possible to activate the outputs at the beginning of the next cycle, after a delay defined by <P5> (step of 30ns and 0 means no delay) |
| | | | | | 6 | Sets / resets several user status bits. The state of the bits is given in parameter <P4> (Bit 0-15: set, bit 16-31: reset) |
| | | | 4 | First parameter associated to the action | | |

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|----------------------------|---------|-----------|--------------|------------------------------------------------------------------|-------------------|------------------------|
| | | | 5 | Second parameter associated to the action | | |
| *IMRES = <P1> | All | 1063 | | Restore the default matrix of the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| *IMROT = <P1>, <P2>, <P3> | All | 1056 | | Rotates the axes plane of the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Rotation plane | 0 | X-Y rotation plane |
| | | | | | 1 | X-Z rotation plane |
| | | | | | 2 | X-theta rotation plane |
| | | | | | 3 | Y-Z rotation plane |
| | | | | | 4 | Y-theta rotation plane |
| | | | | | 5 | Z-theta rotation plane |
| | | | 3 | Rotation angle in micro-degrees | | |
| *IMSCALE = <P1>, ..., <P5> | All | 1055 | | Scales the axes of the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | X axis scale value | | |
| | | | 3 | Y axis scale value | | |
| | | | 4 | Z axis scale value | | |
| | | | 5 | Theta axis scale value | | |
| *IMSHEAR = <P1>, ..., <P5> | All | 1057 | | Shears the defined axis of the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Sheared axis | 0 | X shearing |
| | | | | | 1 | Y shearing |
| | | | | | 2 | Z shearing |
| | | | | | 3 | Theta shearing |
| | | | 3 | Shearing influence 1 | | |
| | | | 4 | Shearing influence 2 | | |
| | | | 5 | Shearing influence 3 | | |
| *IMTRANS = <P1>, ..., <P5> | All | 1054 | | Executes an axes translation of the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | X axis translation value | | |
| | | | 3 | Y axis translation value | | |
| | | | 4 | Z axis translation value | | |
| | | | 5 | Theta axis translation value | | |

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|--------------------------|---------|-----------|--------------|-------------------------------------------------------------------------------------------------|-------------------|-------------------|
| *INCONC = <P1> | All | 1031 | | Quits the concatenated mode in the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| *IND = <P1>[, <P2>] | All | 45 | | Starts a homing sequence on a group of axes | | |
| | | | 1 | Axis mask | | |
| | | | 2 | Disable adjustment movement (disable = 255) | | |
| *IPT = <P1>, ..., <P6> | All | 1045 | | Adds a PT (Position-Time) move to the trajectory for the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | X axis coordinate of the segment end | | |
| | | | 3 | Y axis coordinate of the segment end | | |
| | | | 4 | Z axis coordinate of the segment end | | |
| | | | 5 | Theta axis coordinate of the segment end | | |
| | | | 6 | Time to reach the PT segment end | | |
| *IPVT = <P1>, ..., <P10> | All | 1028 | | Adds a PVT (Position-Velocity-Time) move to the trajectory for the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | X axis coordinate of the segment end | | |
| | | | 3 | Y axis coordinate of the segment end | | |
| | | | 4 | Z axis coordinate of the segment end | | |
| | | | 5 | Theta axis coordinate of the segment end | | |
| | | | 6 | X axis velocity at the end of the segment | | |
| | | | 7 | Y axis velocity at the end of the segment | | |
| | | | 8 | Z axis velocity at the end of the segment | | |
| | | | 9 | Theta axis velocity at the end of the segment | | |
| | | | 10 | Time to reach the PVT segment end | | |
| *IPVTUPDATE = <P1>, <P2> | All | 662 | | Updates the Y registers for PVT trajectory modifications | | |
| | | | 1 | Depth to copy to the depth 0 | | |
| | | | 2 | Mask of the Y registers to copy | | |
| *ISET = <P1>, ..., <P5> | All | 552 | | Defines the nodes associated with the different axes of a specified interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Nodes mask for X axis | | |
| | | | 3 | Nodes mask for Y axis | | |
| | | | 4 | Nodes mask for Z axis | | |
| | | | 5 | Nodes mask for theta axis | | |
| *ISTP = <P1> | All | 656 | | Stops the movement of the specified interpolation group with an infinite deceleration | | |

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|-------------------------------------------|---------|-----------|--------------|-------------------------------------------------------------------------------------|-------------------|-------------------|
| | | | 1 | Interpolation group | | |
| *ITACC = <P1>, <P2> | All | 1036 | | Sets tangential acceleration of the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Tangential acceleration value | | |
| *ITANSPDMASK = <P1>, <P2> | All | 1074 | | Defines which axes are considered for calculating the tangential speed | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Mask of interpolated axes contributing to the tangential speed calculation | | |
| *ITDEC = <P1>, <P2> | All | 1037 | | Sets tangential acceleration of the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Tangential deceleration value | | |
| *ITJRT = <P1>, <P2> | All | 1038 | | Sets tangential jerk time of the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Tangential jerk time | | |
| *ITSPD = <P1>, <P2> | All | 1035 | | Sets tangential speed of the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Tangential speed value | | |
| *IULINE = <P1>, ..., <P5>(<P6>, <P7>) | All | 1033 | | Adds a universal linear move to the trajectory for the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | X axis coordinate of the segment end | | |
| | | | 3 | Y axis coordinate of the segment end | | |
| | | | 4 | Z axis coordinate of the segment end | | |
| | | | 5 | Theta axis coordinate of the segment end | | |
| | | | 6 | Acceleration time | | |
| | | | 7 | Jerk time | | |
| *IUNLOCK = <P1> | All | 655 | | Starts the selected interpolation group by unlocking the 'ILOCK' function | | |
| | | | 1 | Interpolation group | | |
| *IUNOCONC = <P1> | All | 1052 | | Stops before the next universal line for the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| *IURELATIVE = <P1>, <P2> | All | 1051 | | Enables (0) or disables (1) the relative mode of the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Relative mode active or not | | |
| *IUSPDMASK = <P1>, <P2> | All | 1053 | | Sets the axis mask for tangential speed for the selected interpolation group | | |

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|----------------------------|---------|--------|-----------|-----------------------------------------------------------------------------------------------------------------------|----------------|-----------------------------------------------------------------------------------------------------------------|
| | | | 1 | Interpolation group | | |
| | | | 2 | Axis mask for tangential speed | | |
| *IUSPEED = <P1>, <P2> | All | 1049 | | Sets speed for universal linear moves for the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Tangential speed value | | |
| *IUTIME = <P1>, ..., <P3> | All | 1050 | | Sets acceleration and jerk time for universal linear moves for the selected interpolation group | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Acceleration time | | |
| | | | 3 | Jerk time | | |
| *IWTT = <P1>, <P2> | All | 1029 | | Waits for a defined time in interpolated mode (selected interpolation group) | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Waiting time value | | |
| *JMP[.<AXIS>] = <P1>, <P2> | All | 26 | | Start function number <P1> in thread number <P2> | | |
| | | | 1 | Function number | | |
| | | | 2 | Thread number | | |
| *NEW[.<AXIS>] = <P1> | All | 78 | | Restores the default registers values (K), clears registers (X, Y and L) and sequence in ram memory according to <P1> | | |
| | | | 1 | Register selection | 0 | Clears X user variables, Y registers and sequence, sets default values for K parameter in ram memory |
| | | | | | 1 | Clears sequence in ram memory |
| | | | | | 2 | Sets default values for K parameters in ram memory |
| | | | | | 3 | Sets default values for K parameters in ram memory |
| | | | | | 5 | Clears X user variables and Y registers in ram memory |
| | | | | | 6 | Clears L look-up tables in ram memory |
| | | | | | 7 | Clears sequence in ram memory |
| *RCT | All | 754 | | Resets the client TCP/IP communication | | |
| *RES[.<AXIS>] = <P1> | All | 49 | | Restores the flash saved registers (K, X, Y and L) and sequence in ram memory according to <P1> | | |
| | | | 1 | Register selection | 0 | Restores sequence, L look-up tables, K parameters and X user variables and Y registers from flash to ram memory |
| | | | | | 1 | Restores sequence and L look-up tables from flash to ram memory |
| | | | | | 2 | Restores K parameters, X user variables and Y registers from flash to ram memory |

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|-------------------------------|---------|-----------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|--------------------------------------------------------------------------------------------------|
| | | | | | 3 | Restores K parameters from flash to ram memory |
| | | | | | 5 | Restores X user variables and Y registers from flash to ram memory |
| | | | | | 6 | Restores L look-up tables from flash to ram memory |
| | | | | | 7 | Restores sequence from flash to ram memory |
| *RIC | All | 753 | | Reset IO update cycle count (*M709) | | |
| *RST[.<AXIS>] | All | 79 | | Resets the error flag of UltimET light | | |
| RSTDOUT(<P1>) | All | 737 | | Reset a specified digital local output | | |
| | | | 1 | Index of the digital local output to reset | | |
| RSTEDOUT(<P1>) | All | 762 | | Reset a specified digital output | | |
| | | | 1 | Index of the digital output to reset | | |
| *RSU = <P1> | All | 601 | | Perform a reset hardware on the UltimET | | |
| | | | 1 | Security password | | |
| *SAV[.<AXIS>] = <P1> | All | 48 | | Saves the current registers (K, X, Y and L) and sequence in flash memory in function of <P1> | | |
| | | | 1 | Register selection | 0 | Saves sequence, L look-up tables, K parameters, X user variables and Y registers in flash memory |
| | | | | | 1 | Saves sequence and L look-up tables in flash memory |
| | | | | | 2 | Saves K parameters, X user variables and Y registers in flash memory |
| | | | | | 3 | Saves K parameters in flash memory |
| | | | | | 5 | Saves X user variables and Y registers in flash memory |
| | | | | | 6 | Saves L look-up tables in flash memory |
| | | | | | 7 | Saves sequence in flash memory |
| *SETRANGE = <P1>, ..., <P128> | All | 126 | | Fill range of registers at fixed depth starting from register specified as first parameter. For X, XF and L registers, the maximum number of values is 127. For XL and XD registers, the maximum number of values is 100. | | |
| | | | 1 | Type of register, first index and depth | | |
| | | | 2 | Value fixed depth index 1 | | |
| | | | ... | | | |
| | | | 128 | Value fixed depth index 127 | | |
| SETDOUT(<P1>) | All | 736 | | Set a specified digital local output | | |
| | | | 1 | Index of the digital local output to set | | |
| SETEAOUTC(<P1>, <P2>) | All | 731 | | Set a given analog output in converted format | | |
| | | | 1 | Index of the analog output to set | | |

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|-----------------------------------------------------------------------------------------------------|---------|-----------|--------------|--------------------------------------------------------|-------------------|-------------------|
| | | | 2 | Converted value of the analog output | | |
| SETEAOUTR(<P1>, <P2>) | All | 730 | | Set a given analog output in raw format | | |
| | | | 1 | Index of the analog output to set | | |
| | | | 2 | Raw value of the analog output | | |
| SETEDOUT(<P1>) | All | 761 | | Set a specified digital output | | |
| | | | 1 | Index of the digital output to set | | |
| SETEREG(<P1>, <P2>) | All | 725 | | Set a register of the external IO system | | |
| | | | 1 | Address of the register in the Wago controller | | |
| | | | 2 | Value to write in the register | | |
| SETMDOUT(<P1>, <P2>, <P3>) | All | 733 | | Apply a mask on local digital outputs | | |
| | | | 1 | First output of the mask | | |
| | | | 2 | Number of output of the mask | | |
| | | | 3 | Value to put on masked digital output | | |
| SETMEDOUT(<P1>, <P2>, <P3>) | All | 763 | | Apply a mask on external digital outputs | | |
| | | | 1 | First output of the mask | | |
| | | | 2 | Number of output of the mask | | |
| | | | 3 | Value to put on masked digital output | | |
| *SETRTV = <P1>, <P2> | All | 237 | | Sets an RTV slot P1 with a P2 value | | |
| | | | 1 | Slot number | | |
| | | | 2 | Value | | |
| *SSR[.<AXIS>] = <P1>, <P2>, [<P3>, <P4>[, <P5>, <P6>[, <P7>, <P8>[, <P9>, <P10>[, <P11>, <P12>]]]]] | All | 127 | | Set Several Registers 1 up to 6 | | |
| | | | 1 | 1st register and depth definition | | |
| | | | 2 | 1st register value | | |
| | | | 3 | 2nd register and depth definition (optional parameter) | | |
| | | | 4 | 2nd register value (optional parameter) | | |
| | | | 5 | 3th register and depth definition (optional parameter) | | |
| | | | 6 | 3th register value (optional parameter) | | |
| | | | 7 | 4th register and depth definition (optional parameter) | | |
| | | | 8 | 4th register (optional parameter) | | |
| | | | 9 | 5th register and depth definition (optional parameter) | | |
| | | | 10 | 5th register (optional parameter) | | |

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|----------------------------|---------|-----------|--------------|--------------------------------------------------------------------------------------------------------------------------------|-------------------|-------------------|
| | | | 11 | 6th register and depth definition (optional parameter) | | |
| | | | 12 | 6th register (optional parameter) | | |
| *WBC[.<AXIS>] = <P1>, <P2> | All | 54 | | Waits for the specified bits (which have their mask value included in <P2>) of the specified register (<P1>) to be all cleared | | |
| | | | 1 | Value or register to test | | |
| | | | 2 | Bit mask | | |
| *WBS[.<AXIS>] = <P1>, <P2> | All | 55 | | Waits for the specified bits (which have their mask value included in <P2>) of the specified register <P1> to be all set | | |
| | | | 1 | Value or register to test | | |
| | | | 2 | Bit mask | | |
| *WPG[.<AXIS>] = <P1>, <P2> | All | 53 | | Waits until <P1> becomes greater than <P2> | | |
| | | | 1 | Value or register to test | | |
| | | | 2 | Reference value or register | | |
| *WPL[.<AXIS>] = <P1>, <P2> | All | 52 | | Waits until <P1> becomes lower than <P2> | | |
| | | | 1 | Value or register to test | | |
| | | | 2 | Reference value or register | | |
| *WSBC = <P1>, <P2> | All | 515 | | Wait for status (M63) bits cleared in several axes in the same time | | |
| | | | 1 | Bits mask of M63 that must be cleared | | |
| | | | 2 | Mask of the nodes concerned by the wait status bit clear | | |
| *WSBS = <P1>, <P2> | All | 514 | | Wait for status (M63) bits set in several axes in the same time | | |
| | | | 1 | Bits mask of M63 that must be set | | |
| | | | 2 | Mask of the nodes concerned by the wait status bit set | | |
| WTD.<AXIS> = <P1>, <P2> | All | 15 | | Wait all Data are available for the Continuous Traces | | |
| | | | 1 | First index for the range of data | | |
| | | | 2 | Last index for the range of data | | |
| *WTK[.<AXIS>] = <P1>, <P2> | All | 513 | | Waits for the specified mark <P2> of the defined interpolation group <P1> | | |
| | | | 1 | Interpolation group | | |
| | | | 2 | Mark number | | |
| *WTM[.<AXIS>] = <P1> | All | 8 | | Waits for the end of interpolated movement of the group defined by <P1> | | |
| | | | 1 | Interpolation group | | |
| *WTT[.<AXIS>] = <P1> | All | 10 | | Waits for the specified amount of time | | |
| | | | 1 | Time | | |
| *<REG>[.<AXIS>] = <P2> | All | 123 | | Affects the value of <P2> to <REG>. This is available with all kinds of registers excepted monitoring register (M) | | |
| | | | 1 | Destination register | | |

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|--------------------------|---------|-----------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-------------------|
| | | | 2 | Source value or register | | |
| *<REG>[.<AXIS>] += <P2> | All | 91 | | Adds <P2> to <REG> and puts the result in <REG> | | |
| | | | 1 | Destination register | | |
| | | | 2 | Value or register to add | | |
| *<REG>[.<AXIS>] &= <P2> | All | 95 | | Logical AND between each bit of <REG> and <P2> and puts the result in <REG> | | |
| | | | 1 | Destination register | | |
| | | | 2 | Value or register to AND with | | |
| *<REG>[.<AXIS>] &~= <P2> | All | 97 | | Logical NOT AND between each bit of <REG> and <P2> and puts the result in <REG> | | |
| | | | 1 | Destination register | | |
| | | | 2 | Value or register to NOT AND with | | |
| *<REG>[.<AXIS>] = <P2> | All | 122 | | Executes the operation: <REG> = conversion <P2>. This is available with all kinds of registers (<REG> could not be a monitoring register). The syntax is for example *XF2=*X3, or *X3=*XF2 | | |
| | | | 1 | Destination register | | |
| | | | 2 | Source register | | |
| *<REG>[.<AXIS>] /= <P2> | All | 94 | | Divides <REG> by <P2> and puts the result in <REG> | | |
| | | | 1 | Destination register | | |
| | | | 2 | Value or register to divide by | | |
| *<REG>[.<AXIS>] %= <P2> | All | 101 | | Modulo between <REG> and <P2> and puts the result in <REG> | | |
| | | | 1 | Destination register | | |
| | | | 2 | Value or register to modulo with | | |
| *<REG>[.<AXIS>] *= <P2> | All | 93 | | Multiplies <REG> by <P2> and puts the result in <REG> | | |
| | | | 1 | Destination register | | |
| | | | 2 | Value or register to multiply with | | |
| *<REG>[.<AXIS>] ~= <P2> | All | 174 | | Inverts <P2> and puts the result in <REG>. The first operand can only be a K parameter or a X variable, the second one can be an immediate value or any other register. Example: X1.1 ~= M7.1 | | |
| | | | 1 | Destination register | | |
| | | | 2 | Source value or register | | |
| *<REG>[.<AXIS>] = <P2> | All | 96 | | Logical OR between each bit of <REG> and <P2> and puts the result in <REG> | | |
| | | | 1 | Destination register | | |
| | | | 2 | Value or register to OR with | | |
| *<REG>[.<AXIS>] !~= <P2> | All | 98 | | Logical NOT OR between each bit of <REG> and <P2> and puts the result in <REG> | | |
| | | | 1 | Destination register | | |
| | | | 2 | Value or register to NOT OR with | | |

| Syntax | Product | Cmd nb | Param. nb | Command or parameter description | Values or bit# | Value description |
|--------------------------|---------|-----------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|-------------------|
| *<REG>[.<AXIS>] <<= <P2> | All | 173 | | Shifts <REG> of <P2> bit to the left and puts the result in <REG>. The first operand can only be a K parameter or a X variable, the second one can be an immediate value or any other register. Example: X1.1 <<= M7.1 | | |
| | | | 1 | Destination register | | |
| | | | 2 | Value or register to shift by | | |
| *<REG>[.<AXIS>] >>= <P2> | All | 172 | | Shifts <REG> of <P2> bit to the right and puts the result in <REG>. The first operand can only be a K parameter or a X variable, the second one can be an immediate value or any other register. Example: X1.1 >>= M7.1 | | |
| | | | 1 | Destination register | | |
| | | | 2 | Value or register to shift by | | |
| *<REG>[.<AXIS>] -= <P2> | All | 92 | | Subtracts <P2> from <REG> and puts the result in <REG> | | |
| | | | 1 | Destination register | | |
| | | | 2 | Value or register to subtract | | |
| *<REG>[.<AXIS>] ^= <P2> | All | 99 | | Logical XOR between each bit of <REG> and <P2>, puts the result in <REG> | | |
| | | | 1 | Destination register | | |
| | | | 2 | Value or register to XOR with | | |
| *<REG>[.<AXIS>] ^~= <P2> | All | 100 | | Logical NOT XOR between each bit of <REG> and <P2> and puts the result in <REG> | | |
| | | | 1 | Destination register | | |
| | | | 2 | Value or register to NOT XOR with | | |
| *ZTE[.<AXIS>] = <P1> | All | 204 | | With K120=0, starts acquisition if <P1> = 1 (or 2). With K120=1, if <P1>= 1 the axis waits the start acquisition signal, if <P1>=2 the axis becomes the master and sends start acquisition signal. Stops acquisition if <P1>= 0. | | |
| | | | 1 | Enable state | | |

9. Parameters *K

FLOAT_MAX = 3.4028234663852886E+38 and FLOAT_MIN = -3.4028234663852886E+38

| *K | Alias | Product | Values or bit# | Description | Default value | Minimum value | Maximum value |
|-------|-------|---------|----------------|------------------------------------------------------------------------------------------------------------------------------|---------------|---------------|---------------|
| *K33 | | All | | Enables input mode | 0 | 0 | 31 |
| | | | Bit 4 | Enables external error if the DIN match with the mask K305 | | | |
| *K120 | | All | | Enables trace synchronization mode | 0 | 0 | 1 |
| *K121 | | All | | Trace: register selection type (depth 0 for trace 0; depth 1 for trace 1,...) | 3 | 0 | INT32_MAX |
| | | | 1 | User variables register (X) | | | |
| | | | 2 | Parameter register (K) | | | |
| | | | 3 | Monitoring register (M) | | | |
| | | | 8 | Look-up tables register (L) | | | |
| | | | 11 | Advanced user register (Y) | | | |
| | | | 33 | User variables register (XF) | | | |
| | | | 34 | Parameter register (KF) | | | |
| | | | 35 | Monitoring register (MF) | | | |
| | | | 65 | User variables register (XL) | | | |
| | | | 66 | Parameter register (KL) | | | |
| | | | 67 | Monitoring register (ML) | | | |
| | | | 97 | User variables register (XD) | | | |
| | | | 98 | Parameter register (KD) | | | |
| | | | 99 | Monitoring register (MD) | | | |
| *K122 | | All | | Trace: register number (bit#0-15) and depth definition (bit#16-23). Depth 0 for trace 0; depth 1 for trace 1 ,... | 0 | INT32_MIN | INT32_MAX |
| *K123 | | All | | Trace: sampling time selection (period of 25us) | 0 | 0 | INT32_MAX |
| *K124 | | All | | Trace trigger edge selection | 0 | -1 | 1 |
| *K125 | | All | | Trace: trigger mode selection | 0 | 0 | 9 |
| | | | 0 | Immediate acquisition | | | |
| | | | 1 | Start of movement | | | |
| | | | 2 | End of movement | | | |
| | | | 4 | Trigger at a given value (defined in K127 or KF127, KL127, KD127) of the trace defined in K126 | | | |
| | | | 5 | Trigger never starts. Only a ZFT or an external freeze command in synchro mode can stop the acquisition | | | |
| | | | 6 | Trigger at a given value (defined in K127, KF127, KL127 or KD127) of a register defined in K126 | | | |
| | | | 7 | Trigger on a bit field state (K127 or KL127:0: bit to be low, K127 or KL127:1: bit to be high) of a register defined in K126 | | | |

| *K | Alias | Product | Values or bit# | Description | Default value | Minimum value | Maximum value |
|--------|---------|---------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|---------------|---------------|
| | | | 8 | Trigger on a bit field rising or falling edge (K127 or KL127:0: bit on rising edge, K127 or KL127:1: bit on falling edge) of a register defined in K126 | | | |
| | | | 9 | Continuous acquisition mode - No trigger | | | |
| *K126 | | All | | Definition of the trigger parameter according to the trigger mode | 0 | INT32_MIN | INT32_MAX |
| *K127 | | All | | Definition of the trigger level according to the trigger mode | 0 | INT32_MIN | INT32_MAX |
| *KD127 | | All | | Definition of the trigger level according to the trigger mode | 0 | FLOAT_MIN | FLOAT_MAX |
| *KF127 | | All | | Definition of the trigger level according to the trigger mode | 0 | FLOAT_MIN | FLOAT_MAX |
| *KL127 | | All | | Definition of the trigger level according to the trigger mode | 0 | INT64_MIN | INT64_MAX |
| *K128 | | All | | Number of points to acquire for the traces | 1024 | 0 | 16384 |
| *K129 | | All | | Pre or post trigger value | 0 | -16384 | INT32_MAX |
| *K140 | | All | | Error propagation checked group mask | 1 | 0 | 15 |
| | | | Bit 0 | Error propagation group 1 | | | |
| | | | Bit 1 | Error propagation group 2 | | | |
| | | | Bit 2 | Error propagation group 3 | | | |
| | | | Bit 3 | Error propagation group 4 | | | |
| *K141 | | All | | Error propagation published group mask | 1 | 0 | 15 |
| | | | Bit 0 | Error propagation group 1 | | | |
| | | | Bit 1 | Error propagation group 2 | | | |
| | | | Bit 2 | Error propagation group 3 | | | |
| | | | Bit 3 | Error propagation group 4 | | | |
| *K142 | | All | | Error routine definition. K142:thread.<axis>=<error_routine>. If K142 = -1: thread 1 jumps to the routine called errhandler if it exists, otherwise the thread is stopped; thread 2 or 3 is stopped. If K142 = -2: the thread is not stopped. | -1 | -2 | 1023 |
| *K143 | | All | | Sequence management: depth of the local variable associated with each thread (*K143:depth define the concerned thread, 0 is not used) | 0 | 0 | 3 |
| *K144 | | All | | Thread not stopped by HLT HLB HLO. Bit#1 for thread 1, bit#2 for thread 2., bit#3 for thread 3 | 0 | 0 | 14 |
| *K166 | WARNMSK | All | | Warning mask | 0 | 0 | UINT32_MAX |
| | | | Bit 19 | Sequence debugger: breakpoints present. | | | |
| | | | Bit 20 | Interpolation buffer nearly full. | | | |
| *K171 | DOUT | All | | Digital outputs | 0 | 0 | INT32_MAX |
| *K177 | | All | | User status | 0 | INT32_MIN | INT32_MAX |
| *K198 | | All | | The register number is given by K198 when it is equal to Y or y (ex: Ky.1=...) | 0 | 0 | UINT16_MAX |
| *K200 | | All | | Time allowed for compiled sequence | 0 | 0 | 13333 |
| *K297 | | All | | Mask for active thread information on M63 status (bit#1 thread1, bit#2 thread2, bit#3 thread3) | 0 | 0 | 14 |

| *K | Alias | Product | Values or bit# | Description | Default value | Minimum value | Maximum value |
|-------|------------|-----------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|---------------|---------------|
| *K305 | | All | | Requested DIN mask for external error (bit 0-15: state 1, bit 16-31: state 0) | 0 | 0 | 1048576 |
| *K353 | | All | | DOUT mask for TransnET cycle | 0 | 0 | 15 |
| *K357 | | All | | DOUT mask for security management (bit 0-15: state 1, bit 16-31: state 0) | 0 | 0 | 1048576 |
| *K360 | | All | | Allows to output on the DOUT one TransnET synchro signal every K360 cycles (0=no signal, 1=each TransnET cycle, 2= every 2 TransnET cycle,...) | 0 | 0 | UINT16_MAX |
| *K415 | | PCI, PCIe | | Enable fast stack for Compiled Sequences. Warning: the stack size is significantly reduced | 0 | 0 | 1 |
| | | | 0 | Sequence stack in slow external memory (2048 32-bit integers) | | | |
| | | | 1 | Sequence stack in fast internal memory (128 32-bit integers) | | | |
| *K513 | IBUFWNG | All | | Interpolation: threshold of the percent of command buffer size of remaining free slots that generates a warning | 30 | 0 | 100 |
| *K522 | | All | | Interpolation, unit position factor. The depths correspond to the ipol group numbers | 1 | 1 | INT32_MAX |
| *K523 | | All | | Interpolation, unit position exponent. The depths correspond to the ipol group numbers | -8 | -16 | 0 |
| *K524 | | All | | Interpolation, unit speed exponent. The depths correspond to the ipol group numbers | -6 | -10 | 0 |
| *K525 | | All | | Interpolation, unit acceleration exponent. The depths correspond to the ipol group numbers | -6 | -10 | 0 |
| *K526 | | All | | Interpolation, unit time exponent. The depths correspond to the ipol group numbers | -6 | -10 | 0 |
| *K530 | ISPD RATE | All | | Interpolation, speed rate (unit: %). The depths correspond to the ipol group numbers | 100 | 0 | 100 |
| *K532 | | All | | Interpolation, acc. time for break with IULINE. The depths correspond to the ipol group numbers | 0 | 0 | INT32_MAX |
| *K533 | | All | | Interpolation, jerk time for break with IULINE. The depths correspond to the ipol group numbers | 0 | 0 | INT32_MAX |
| *K540 | | All | | Interpolation, mark number. The depths correspond to the ipol group numbers | 0 | 0 | INT32_MAX |
| *K541 | | All | | Interpolation, DOUT reservation for the marks | 0 | 0 | 15 |
| *K545 | | All | | Interpolation, ULM speed limit for each axis (depths 0 = axis X of group 0, 1=Y of group 0, ..., 4=X of group1, ...) | 0 | 0 | INT32_MAX |
| *K546 | | All | | Interpolation, deceleration value for each axis in case of emergency braking in PVT mode (depths 0 = axis X of group 0, 1=Y of group 0, ..., 4=X of group1, ...) | 0 | 0 | INT32_MAX |
| *K560 | | All | | Linked to IPV TUPDATE command: stores the first number of a block of 32 Y registers that can be updated | 0 | 0 | 224 |
| *K561 | | All | | Time value used for a speed rate change from 100% to 0% along the trajectory in PVT and PT mode. The depths correspond to the ipol group numbers | 1 | 1 | INT32_MAX |
| *K580 | | All | | Interpolation, User conversion factor to fix a ratio between interpolated axes: activation parameter (enable = 1 and disable = 0, depth 0 = group 0, depth 1 = group 1) | 0 | 0 | 1 |
| *K581 | | All | | Interpolation, User conversion factor to fix a ratio between interpolated axes: numerator (depths correspond to interpolated axes 0=X group 0, 1=Y group 0,...) | 1 | 1 | INT32_MAX |
| *K582 | | All | | Interpolation, User conversion factor to fix a ratio between interpolated axes: denominator (depths correspond to interpolated axes 0=X group 0, 1=Y group 0,...) | 1 | 1 | INT32_MAX |
| *K595 | LED | All | | Control of user's LED (1=on, 0=off. Depth 0 = green led, depth 1 = red led) | 0 | 0 | INT32_MAX |
| *K600 | | All | | Sets the interrupt refresh frequency for interpolation | 0 | 0 | 16 |
| *K610 | EAOUTNBIT0 | All | | External IO, N bits analog outputs (0-7) | 0 | 1 | 32 |
| *K611 | EAOUTNBIT1 | All | | External IO, N bits analog outputs (8-15) | 0 | 1 | 32 |

| *K | Alias | Product | Values or bit# | Description | Default value | Minimum value | Maximum value |
|--------|------------|---------|-------------------|--------------------------------------------------------------------------------------|------------------|------------------|------------------|
| *K615 | EAINNBIT0 | All | | External IO, N bits analog inputs (0-7) | 0 | 1 | 32 |
| *K616 | EAINNBIT1 | All | | External IO, N bits analog inputs (8-15) | 0 | 1 | 32 |
| *K620 | EAOUTOFF0 | All | | External IO, offset analog outputs (0-7) | 0 | INT32_MIN | INT32_MAX |
| *KF620 | EAOUTGAIN0 | All | | External IO, analog output gains (0-7) | 1 | FLOAT_MIN | FLOAT_MAX |
| *K621 | EAOUTOFF1 | All | | External IO, offset analog outputs (8-15) | 0 | INT32_MIN | INT32_MAX |
| *KF621 | EAOUTGAIN1 | All | | External IO, analog output gains (8-15) | 1 | FLOAT_MIN | FLOAT_MAX |
| *K625 | EAINOFF0 | All | | External IO, offset analog inputs (0-7) | 0 | INT32_MIN | INT32_MAX |
| *KF625 | EAINGAIN0 | All | | External IO, analog input gains (0-7) | 1 | FLOAT_MIN | FLOAT_MAX |
| *K626 | EAINOFF1 | All | | External IO, offset analog inputs (8-15) | 0 | INT32_MIN | INT32_MAX |
| *KF626 | EAINGAIN1 | All | | External IO, analog input gains (8-15) | 1 | FLOAT_MIN | FLOAT_MAX |
| *KL630 | | All | | Mask of the nodes to monitor for node error detection | 0 | 0 | INT64_MAX |
| *K631 | | All | | Action to take in case of error | 0 | 0 | INT32_MAX |
| *KL632 | | All | | In case of error, mask of the nodes which action is applied to | 0 | 0 | INT64_MAX |
| *K717 | ICAM | All | | Interpolation, came value (unit: %). The depths correspond to the ipol group numbers | 100 | 1 | 100 |
| *K721 | IPCLADR | All | | External IO, IP address of the UltimET TCPIP client | 0 | INT32_MIN | INT32_MAX |
| *K722 | IOIPADR | All | | External IO, external IO module server IP address | 0 | INT32_MIN | INT32_MAX |
| *K724 | IOTIMEOUT | All | | External IO, external IO module response timeout (ms) | 1000 | 0 | 9999 |
| *K725 | IOSUBNET | All | | External IO, subnet mask used by the TCPIP external IO module | 0 | INT32_MIN | INT32_MAX |
| *K726 | IOGATEWY | All | | External IO, external IO default gateway IP address | 0 | INT32_MIN | INT32_MAX |
| *K727 | EIOEN | All | | External IO, Feature enable | 0 | 0 | 1 |
| *K728 | IOREOPNT | All | | External IO, delay before retrying to open communication (s) | 2 | 0 | 9999 |
| *K729 | IOREFRESH | All | | External IO, external IO refresh rate (ms) | 10 | 0 | 9999 |
| *K734 | IONBREP | All | | External IO, number of repetition of UDP message before error | 1 | 0 | 5 |
| *K740 | EAOUTR0 | All | | External IO, set value for raw analog outputs (0-7) | 0 | INT32_MIN | INT32_MAX |
| *KF740 | EAOUTC0 | All | | External IO, set value for converted analog outputs (0-7) | 0 | FLOAT_MIN | FLOAT_MAX |
| *K741 | EAOUTR1 | All | | External IO, set value for raw analog outputs (8-15) | 0 | INT32_MIN | INT32_MAX |
| *KF741 | EAOUTC1 | All | | External IO, set value for converted analog outputs (8-15) | 0 | FLOAT_MIN | FLOAT_MAX |
| *K760 | EDOUT0 | All | | External IO, set value for digital outputs (0-127, 16 bits per depth) | 0 | 0 | UINT16_MAX |

10. Monitorings *M

| *M | Alias | Product | Values or bit# | Description |
|-------|---------|---------|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *M50 | DIN | All | | Digital inputs |
| *M63 | SDC | All | | UltimET light status |
| | | | Bit 0 | User word bit 0, could be modified by parameter K177 |
| | | | Bit 1 | User word bit 1, could be modified by parameter K177 |
| | | | Bit 2 | User word bit 2, could be modified by parameter K177 |
| | | | Bit 3 | User word bit 3, could be modified by parameter K177 |
| | | | Bit 4 | User word bit 4, could be modified by parameter K177 |
| | | | Bit 5 | User word bit 5, could be modified by parameter K177 |
| | | | Bit 6 | User word bit 6, could be modified by parameter K177 |
| | | | Bit 7 | User word bit 7, could be modified by parameter K177 |
| | | | Bit 8 | User word bit 8, could be modified by parameter K177 |
| | | | Bit 9 | User word bit 9, could be modified by parameter K177 |
| | | | Bit 10 | User word bit 10, could be modified by parameter K177 |
| | | | Bit 11 | User word bit 11, could be modified by parameter K177 |
| | | | Bit 12 | User word bit 12, could be modified by parameter K177 |
| | | | Bit 13 | User word bit 13, could be modified by parameter K177 |
| | | | Bit 14 | User word bit 14, could be modified by parameter K177 |
| | | | Bit 15 | User word bit 15, could be modified by parameter K177 |
| | | | Bit 19 | Bit present, always 1 |
| | | | Bit 20 | UltimET light is executing an interpolated trajectory |
| | | | Bit 23 | UltimET light is in warning mode |
| | | | Bit 24 | UltimET light is executing an internal sequence |
| | | | Bit 25 | UltimET light is executing an interpolated trajectory, group 0 |
| | | | Bit 26 | UltimET light is in error mode |
| | | | Bit 27 | Trace busy flag is set during a register trace acquisition |
| | | | Bit 29 | UltimET light is executing an interpolated trajectory, group 1 |
| | | | Bit 30 | TransnET connection has been interrupted |
| | | | Bit 31 | The controller is executing thread depending setting *K297 |
| *M64 | ERRCD | All | | Gives the error code |
| *M66 | WARCD | All | | Warning code |
| *M70 | CTYP | All | | Indicates the type of UltimET light |
| *M71 | | All | | Gives the software boot version of UltimET light. A special ETEL procedure allows the conversion of this value in the software boot version (format is similar to M72) |
| *M72 | VER | All | | Gives the firmware version of the UltimET light. A special ETEL procedure allows the |
| *M73 | SER | All | | Gives the serial number of the UltimET light |
| *M85 | ARTICLE | All | | Gives the article number string. The 18 strings of the article number are read using 5 depths of M85. Each depth shows 4 strings (in ASCII) |
| *M87 | | All | | Gives the UltimET axis number |
| *M97 | | All | | Mask of the sequence threads currently active |
| *M101 | | All | | Mask of the threads currently in waiting state |
| | | | Bit 1 | Sequence thread 1 |
| | | | Bit 2 | Sequence thread 2 |
| | | | Bit 3 | Sequence thread 3 |
| | | | Bit 16 | Application bus PCI |
| | | | Bit 17 | Application bus TCP/IP Port 1 |

| *M | Alias | Product | Values or bit# | Description |
|--------|-------|---------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| | | | Bit 18 | Application bus TCP/IP Port 2 |
| | | | Bit 19 | Application bus TCP/IP Port 3 |
| *M120 | | All | | Traces synchronization mode |
| *M121 | | All | | Trace register selection type (depth 0 for trace 0; depth 1 for trace 1,...) |
| | | | 1 | User variables register (X) |
| | | | 2 | Parameter register (K) |
| | | | 3 | Monitoring register (M) |
| | | | 8 | Look-up tables register (L) |
| | | | 11 | Advanced user register (Y) |
| | | | 33 | User variables register (XF) |
| | | | 34 | Parameter register (KF) |
| | | | 35 | Monitoring register (MF) |
| | | | 65 | User variables register (XL) |
| | | | 66 | Parameter register (KL) |
| | | | 67 | Monitoring register (ML) |
| | | | 97 | User variables register (XD) |
| | | | 98 | Parameter register (KD) |
| | | | 99 | Monitoring register (MD) |
| *M122 | | All | | Trace register number and depth selection (bit#0-15 for register number, bit#16-23 for depth) |
| *M123 | | All | | Trace sampling time (period of 25us) |
| *M124 | | All | | Trace trigger edge |
| *M125 | | All | | Trace trigger mode |
| | | | 0 | Immediate acquisition |
| | | | 1 | Start of movement |
| | | | 2 | End of movement |
| | | | 4 | Trigger at a given value (defined in K127 or KF127, KL127, KD127) of the trace defined in K126 |
| | | | 5 | Trigger never starts. Only a ZFT or an external freeze command in synchro mode can stop the acquisition |
| | | | 6 | Trigger at a given value (defined in K127, KF127, KL127 or KD127) of a register defined in K126 |
| | | | 7 | Trigger on a bit field state (K127 or KL127:0: bit to be low, K127 or KL127:1: bit to be high) of a register defined in K126 |
| | | | 8 | Trigger on a bit field rising or falling edge (K127 or KL127:0: bit on rising edge, K127 or KL127:1: bit on falling edge) of a register defined in K126 |
| | | | 9 | Continuous acquisition mode - No trigger |
| *M126 | | All | | Definition of the trigger parameter according to the trigger mode |
| *M127 | | All | | The value gives the trigger level according to the trigger mode |
| *MD127 | | All | | The value gives the trigger level according to the trigger mode |
| *MF127 | | All | | The value gives the trigger level according to the trigger mode |
| *ML127 | | All | | The value gives the trigger level according to the trigger mode |
| *M128 | | All | | Number of acquired points for the trace |
| *M129 | | All | | Pre or post trigger value |
| *M131 | | All | | Traces acquisition state |
| | | | 0 | Acquisition state: Initialization 1 |
| | | | 1 | Acquisition state: Initialization 2 |
| | | | 2 | Acquisition state: Wait for buffer synchronization |
| | | | 3 | Acquisition state: Wait for trigger |
| | | | 4 | Acquisition state: Trigger condition reached |

| *M | Alias | Product | Values or bit# | Description |
|--------|---------|-----------|----------------|------------------------------------------------------------------------------------------------------------------------------|
| | | | 5 | Acquisition state: End of acquisition |
| *M134 | | All | | Freezes time of the measure |
| *ML134 | | All | | Freezes time of the measure |
| *M135 | | All | | Number of the T register with the trigger condition |
| *M136 | | All | | Number of available points |
| *M141 | | All | | TransnET error groups status |
| *M166 | WARNFLG | All | | Warning flags |
| | | | Bit 19 | Sequence debugger: breakpoints present. |
| | | | Bit 20 | Interpolation buffer nearly full. |
| *M171 | | All | | Gives the state of the digital outputs (reflects parameter *K171) |
| *M201 | | All | | Time quota for thread 1 (depth 1), thread 2 (depth 2), and thread 3 (depth 3) |
| *M202 | | All | | Time used by thread 1 (depth 1), thread 2 (depth 2), and thread 3 (depth 3) |
| *M203 | | All | | UltimET board, TCPIP server MAC address |
| *M228 | | All | | Reference time counter in unit of 25us. This counter runs freely and is synchronized with UltimET when TransnET is connected |
| *ML228 | | All | | Reference time counter in unit of 25us. This counter runs freely and is synchronized with UltimET when TransnET is connected |
| *M282 | | All | | Registers or sequence version (depth 0 to 2: not used, depth 3 to 7 as for SAV command) |
| *M417 | | PCI, PCIe | | Sequence stack size in 32-bit words |
| *M418 | | PCI, PCIe | | Sequence stack free space in 32-bit words |
| *M443 | | All | | Number of possible RTV write slots (ASW) |
| *M447 | | All | | Number of possible RTV read slots (ASR) |
| *M448 | | All | | Used RTV slots (by ASW) |
| *M449 | | All | | Number of written RTV (by ASW command) |
| *M450 | | All | | RTV monitoring M450 |
| *MD450 | | All | | RTV monitoring MD450 |
| *MF450 | | All | | RTV monitoring MF450 |
| *ML450 | | All | | RTV monitoring ML450 |
| *M451 | | All | | RTV monitoring M451 |
| *MD451 | | All | | RTV monitoring MD451 |
| *MF451 | | All | | RTV monitoring MF451 |
| *ML451 | | All | | RTV monitoring ML451 |
| *M452 | | All | | RTV monitoring M452 |
| *MD452 | | All | | RTV monitoring MD452 |
| *MF452 | | All | | RTV monitoring MF452 |
| *ML452 | | All | | RTV monitoring ML452 |
| *M453 | | All | | RTV monitoring M453 |
| *MD453 | | All | | RTV monitoring MD453 |
| *MF453 | | All | | RTV monitoring MF453 |
| *ML453 | | All | | RTV monitoring ML453 |
| *M454 | | All | | RTV monitoring M454 |
| *MD454 | | All | | RTV monitoring MD454 |
| *MF454 | | All | | RTV monitoring MF454 |
| *ML454 | | All | | RTV monitoring ML454 |
| *M455 | | All | | RTV monitoring M455 |
| *MD455 | | All | | RTV monitoring MD455 |

| *M | Alias | Product | Values or bit# | Description |
|--------|----------|---------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| *MF455 | | All | | RTV monitoring MF455 |
| *ML455 | | All | | RTV monitoring ML455 |
| *M456 | | All | | RTV monitoring M456 |
| *MD456 | | All | | RTV monitoring MD456 |
| *MF456 | | All | | RTV monitoring MF456 |
| *ML456 | | All | | RTV monitoring ML456 |
| *M457 | | All | | RTV monitoring M457 |
| *MD457 | | All | | RTV monitoring MD457 |
| *MF457 | | All | | RTV monitoring MF457 |
| *ML457 | | All | | RTV monitoring ML457 |
| *M458 | | All | | RTV monitoring M458 |
| *MF458 | | All | | RTV monitoring MF458 |
| *M459 | | All | | RTV monitoring M459 |
| *MF459 | | All | | RTV monitoring MF459 |
| *M460 | | All | | RTV monitoring M460 |
| *MF460 | | All | | RTV monitoring MF460 |
| *M461 | | All | | RTV monitoring M461 |
| *MF461 | | All | | RTV monitoring MF461 |
| *M462 | | All | | RTV monitoring M462 |
| *MF462 | | All | | RTV monitoring MF462 |
| *M463 | | All | | RTV monitoring M463 |
| *MF463 | | All | | RTV monitoring MF463 |
| *M464 | | All | | RTV monitoring M464 |
| *MF464 | | All | | RTV monitoring MF464 |
| *M465 | | All | | RTV monitoring M465 |
| *MF465 | | All | | RTV monitoring MF465 |
| *M470 | | All | | Maximum downloadable sharc instructions for a compiled sequence |
| *M471 | | All | | Maximum downloadable data longs for a compiled sequence |
| *M472 | | All | | Maximum downloadable source characters for a compiled sequence |
| *M475 | | All | | Effectively downloaded sharc instructions in the last compiled sequence |
| *M476 | | All | | Effectively downloaded data longs in the last compiled sequence |
| *M477 | | All | | Effectively downloaded source characters in the last compiled sequence |
| *M512 | INBUFFRE | All | | Interpolation: remaining free slots in interpolation commands buffer |
| *ML512 | | All | | Mask of the nodes present on the TransnET |
| *M513 | IBUFNMIN | All | | Interpolation: actual threshold of the number of remaining free slots in command buffer that generates a warning |
| *M516 | | All | | Gives the gate number associated to each thread (depth 0 = application bus, depth 1 = thread 1...) |
| *M518 | | All | | Interpolation group moving state. lpol group 0 moving state corresponds to bit 0 (1 = moving), lpol group 1 moving state corresponds to bit 1 |
| *M519 | | All | | Interpolation status (depth 0=group 0...). |
| | | | Bit 0 | Active state. |
| | | | Bit 1 | Moving state. |
| | | | Bit 2 | Emergency brake state. |
| *M520 | | All | | Copy of the M63 of all the nodes present on TransnET. The depths correspond to the node number |
| *M524 | | All | | Number of free RTV0 slots. These RTV0 slots can be managed using, get or free slot commands |
| *M525 | | All | | First RTV0 slot number |

| *M | Alias | Product | Values or bit# | Description |
|--------|------------|---------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *ML525 | | All | | Interpolation, interpolated axis mask (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M526 | | All | | Gives the number of RTV0 slots |
| *M530 | | All | | Interpolation group is active (1) or not (0). The depths correspond to the ipol group numbers |
| *M531 | | All | | Interpolation buffer locked (1) or not (0). The depths correspond to the ipol group numbers |
| *M535 | | All | | Number of commands inside interpolation buffer. The depths correspond to the ipol group numbers |
| *M590 | | All | | Interpolation, absolute coordinates mode: absolute (1) or relative (0). The depths correspond to the ipol group numbers |
| *M629 | | All | | TransnET frame error counter |
| *ML630 | | All | | Mask of all the error bits of the nodes present on the TransnET |
| *M640 | | All | | User position of X axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M641 | | All | | Real position of X axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M642 | | All | | User position of Y axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M643 | | All | | Real position of Y axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M644 | | All | | User position of Z axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M645 | | All | | Real position of Z axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M646 | | All | | User position of THETA axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M647 | | All | | Real position of THETA axis, ipol group 0 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M649 | | All | | Tangential velocity, ipol group 0 (unit: ufsi) |
| *M650 | | All | | Tangential acceleration, ipol group 0 (unit: ufai) |
| *M651 | | All | | Tangential deceleration, ipol group 0 (unit: ufai) |
| *M652 | | All | | Tangential jerk time, ipol group 0 (unit: ufti) |
| *M658 | | All | | Maximal time value for interpolation commands (unit: ufti) |
| *M659 | | All | | Maximal distance for interpolation commands (unit: ufpi) |
| *M660 | | All | | User position of X axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M661 | | All | | Real position of X axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M662 | | All | | User position of Y axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M663 | | All | | Real position of Y axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M664 | | All | | User position of Z axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M665 | | All | | Real position of Z axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M666 | | All | | User position of THETA axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M667 | | All | | Real position of THETA axis, ipol group 1 (unit: incr, refer to the controllers associated to this axis for the conversion) |
| *M669 | | All | | Tangential velocity, ipol group 1 (unit: ufsi) |
| *M670 | | All | | Tangential acceleration, ipol group 1 (unit: ufai) |
| *M671 | | All | | Tangential deceleration, ipol group 1 (unit: ufai) |
| *M672 | | All | | Tangential jerk time, ipol group 1 (unit: ufti) |
| *M680 | TANSPD | All | | Current interpolation tangential speed. The depth (0 and 1) correspond to the interpolation group number |
| *MF680 | TANSPEED | All | | Current interpolation tangential speed. The depth (0 and 1) correspond to the interpolation group number |
| *M681 | TANSPDMASK | All | | Tangential speed is calculated based on axes present in this mask. Current interpolation tangential speed. The depth (0 and 1) correspond to the interpolation group number |

| *M | Alias | Product | Values or bit# | Description |
|--------|-----------|---------|----------------|----------------------------------------------------------------------------------------------------------------------|
| *M690 | NEAOUT | All | | External IO, number of analog outputs |
| *M691 | NEAIN | All | | External IO, number of analog inputs |
| *M692 | NEDOUT | All | | External IO, number of digital outputs |
| *M693 | NEDIN | All | | External IO, number of digital inputs |
| *M694 | IOCPLREF | All | | External IO, converted coupler reference |
| *M695 | IOMODREF | All | | External IO, IO modules references |
| *M696 | IOEXC | All | | External IO, IO modbus exception registers |
| *M697 | IOEXCSRC | All | | External IO, IO modbus exception source |
| | | | 10 | Error getting the number of analog inputs |
| | | | 20 | Error getting the number of analog outputs |
| | | | 30 | Error getting the number of digital inputs |
| | | | 40 | Error getting the number of digital outputs |
| | | | 50 | Error getting the coupler reference |
| | | | 60 | Error getting the module references |
| | | | 70 | Error updating the external inputs/outputs |
| | | | 90 | Error getting the value of a register in the Wago system |
| | | | 95 | Error setting the value of a register in the Wago system |
| *M698 | IOENBLD | All | | External IO, feature enabled status |
| *M699 | NMACS | All | | UltimET board, number of active MACs |
| *M705 | IOSTS | All | | External IO, data exchange activity |
| | | | 10 | Not active |
| | | | 20 | Initializing |
| | | | 25 | IO read cycle |
| | | | 30 | IO update cycle |
| | | | 40 | In error |
| *M706 | IOMXRETIM | All | | External IO, maximum elapsed time between updates |
| *M707 | IOMACADR | All | | External IO, TCP/IP client MAC address |
| *M708 | IORETIM | All | | External IO, elapsed time since last update |
| *M709 | IONCYCLE | All | | External IO, Data exchange cycle count |
| *M712 | IONLOST | All | | External IO, total number of lost message |
| *M726 | | All | | Interpolation, axis present (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M727 | | All | | Interpolation, number of axes per ipol axis (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M728 | | All | | Interpolation, axis encoder ipol factor (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M729 | | All | | Interpolation, axis K77 ipol factor (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M730 | | All | | Interpolation, axis motor type (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M731 | | All | | Interpolation, axis encoder period (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M732 | | All | | Interpolation, axis motor mult. factor (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M733 | | All | | Interpolation, axis motor div. factor (depths 0 = axis X group 0, 1=Y group 0..., depth 4 = axis X group 1...) |
| *M740 | EAOUTRS0 | All | | External IO, state of raw analog outputs (0-7) |
| *MF740 | EAOUTCS0 | All | | External IO, state of converted analog outputs (0-7) |
| *M741 | EAOUTRS1 | All | | External IO, state of raw analog outputs (8-15) |
| *MF741 | EAOUTCS1 | All | | External IO, state of converted analog outputs (8-15) |
| *M745 | EAINRS0 | All | | External IO, state of raw analog inputs (0-7) |

| *M | Alias | Product | Values or bit# | Description |
|--------|----------|---------|----------------|-----------------------------------------------------------------------|
| *MF745 | EAINCS0 | All | | External IO, state of converted analog inputs (0-7) |
| *M746 | EAINRS1 | All | | External IO, state of raw analog inputs (8-15) |
| *MF746 | EAINCS1 | All | | External IO, state of converted analog inputs (8-15) |
| *M760 | EDOUTS0 | All | | External IO, state out the digital outputs (0-127, 16 bits per depth) |
| *M764 | ISEIOSTA | All | | External IO, update cycle status |
| | | | 0 | Writing of the IOs is disabled, the state of the IOs is read |
| | | | 1 | Cyclic reading and writing of the IOs is active |
| *M765 | EDINS0 | All | | External IO, digital inputs (0-127, 16 inputs per depth) |

11. Warning reference list

This error code is given by the monitoring *M66.

| *M66 | Description | Solutions | Product |
|------|----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|---------|
| 18 | Interpolation: emergency brake has occurred. | General case: - An emergency brake has occurred, because the velocity of the last PVT segment was not zero. | All |
| 19 | Sequence debugger: breakpoints present. | General case: - The user is in debugger mode on the ComET editor. | All |
| 20 | Interpolation buffer nearly full. | General case: - Use iLock/iUnlock method to regulate the buffer of the interpolation. | All |
| 25 | High priority interrupt overflow. | General case: - The UltimET is overloaded, if you have two interpolated groups, use *k600 to desynchronize both groups. | All |
| 30 | Trace acquisition has overrun upload. | General case: - The Traces upload is not fast enough and data was corrupted. Increase the upload rate or reduce sampling frequency. | All |

12. Errors reference list

This error code is given by the monitoring *M64.

| *M64 | Description | Solutions | Product |
|------|--------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|---------|
| 401 | Sequence: bad register index or depth | General case: - Sequence: bad register index or depth. | All |
| 402 | Sequence: bad thread | General case: - A non-existent thread has been called from the sequence; the limitation is 2 on the AccurET and 3 on UltimET. | All |
| 403 | Sequence: thread already running | General case: - Sequence: Thread already running. | All |
| 404 | Sequence: division by zero | General case: - Sequence: Division by zero. | All |
| 405 | Sequence: bad command parameter | General case: - Sequence: Bad command parameter. | All |
| 406 | Sequence: bad register access | General case: - Sequence: Bad register access. | All |
| 407 | Sequence: bad node | General case: - Sequence: Bad node. | All |
| 408 | Sequence: call to a function that is not implemented in the firmware | General case: - Sequence: function not existing. | All |
| 460 | Sequence: incorrect sequence format in flash | General case: - Sequence : incorrect sequence format in flash. | All |
| 1000 | Communication error. TransnET down | General case: - Communication error: The bus has not been initialized properly. | All |
| 1001 | Communication error. Time-out during the request of a monitoring channel on the TransnET. | General case: - Communication error: Time-out during the request of a monitoring channel on the TransnET. | All |
| 1002 | Communication error. Time-out waiting for the controller's answer to a monitoring request. | General case: - Communication error: Time-out waiting for the controller's answer to a monitoring request. | All |

| *M64 | Description | Solutions | Product |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| 1003 | Communication error. The destination node of the monitoring request is not present. | General case: - Communication error: The destination node of the monitoring request is not present. | All |
| 1004 | Communication error. The destination node has disappeared. | General case: - Communication error: The destination node has disappeared. | All |
| 1005 | Communication error. The bus has not been initialized properly | General case: - Communication error: The bus has not been initialized properly. | All |
| 1006 | Communication error. The nodes' statuses are not properly configured on the bus. | General case: - Communication error: The node's statuses are not properly configured on the bus. | All |
| 1008 | Communication error. There are more nodes than expected on the bus. | General case: - Communication error: There are more nodes than expected on the bus. | All |
| 1009 | Communication error: too many TransnET buffer switches when reading 64-bit monitoring. | General case: - Communication error: Too many transnET buffer switches when reading 64-bit monitoring. | All |
| 1010 | Communication error. One AccurET controller firmware is not compatible with UltimET firmware. AccurET firmware must be greater or equal to 2.00 | General case: - Communication error: One AccurET controller firmware is not compatible with UltimET firmware. AccurET firmware must be greater or equal to 2.00. | All |
| 1012 | Communication error. The buffer of statuses received from transnET for a given axis is full: possible loss of statuses for the application | General case: - Communication error: The buffer of statuses received from TransnET for a given axis is full: possible loss of statuses for the application. | All |
| 1013 | Communication error: TransnET timeout waiting for nodes | General case: - Communication error: TransnET timeout waiting for nodes. | All |
| 1014 | Communication error: inconsistent number of slaves (ML512 and HSCI) | General case: - Communication error: Inconsistent number of slaves (ML512 and HSCI) | All |
| 1015 | Communication error. Error collecting nodes' data at bus initialization | General case: - Communication error: Error collecting node's data at bus initialization. | All |
| 1020 | Communication: the destination node of the urgent command is not present | General case: - Communication error: The destination node of the urgent command is not present. | All |
| 1021 | Communication: the destination node of the urgent command disappeared | General case: - Communication error: The destination node of the urgent command has disappeared. | All |
| 1022 | Communication: timeout while sending an urgent command on a TransnET channel | General case: - Communication error: Time-out during the send of an urgent command on a TransnET channel. | All |
| 1023 | Communication: timeout while waiting for the answer to an urgent command from the controller | General case: - Communication error: Time-out during the wait of the controller's answer to an urgent command. | All |
| 1031 | Communication error. The normal command destination node has disappeared | General case: - Communication error: The normal command destination node has disappeared. | All |
| 1032 | Command error: an error in the execution of a local command has precluded its acknowledgment | General case: - Command Error: An error in the execution of a local command has precluded its acknowledgment. | All |
| 1200 | RTV management. Slot has not been get | General case: - RTV management: Slot has not been get | All |
| 1201 | RTV management. Slot out of range | General case: - RTV management: Slot out of range | All |
| 1202 | RTV management. Slot already in use | General case: - RTV management: Slot already in use | All |
| 1210 | RTV management. Bad parameter in the ASW command | General case: - RTV management: Bad parameter in the ASW command. | All |
| 1211 | RTV management. ASW slot already assigned | General case: - RTV management: ASW slot already assigned. | All |
| 1220 | RTV management. Bad parameter in the ASR command | General case: - RTV management: Bad parameter in the ASR command. | All |
| 1230 | RTV management. Bad parameter in SETRTV command | General case: - RTV management: Bad parameter in SETRTV command. | All |
| 1500 | Stream error. No access to the buffer 0x2E on the PCI. Impossible to reply to the PC | General case: - Stream error: No access to the buffer 0x2E on the PCI. Impossible to reply to the PC. | All |
| 1501 | Stream error. Error in the received frame 0x2E | General case: - Stream error: Error in the received frame 0x2E. | All |
| 1502 | Stream error. Error in the generated frame 0x2E (internal error) | General case: - Stream error: Error in the generated frame 0x2E (internal error) | All |
| 1503 | Stream error. Error during the writing of a chunk in RAM | General case: - Stream error: Error during the writing of a chunk in RAM. | All |
| 1504 | Stream error. A command to start the stream is received while another one is already running | General case: - Stream error: A command to start the stream is received while another one is already running. | All |

| *M64 | Description | Solutions | Product |
|------|--------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|---------|
| 1505 | Stream error. Error if an axis does not get a command for the stream | General case: - Stream error: Error if an axis does not get a command for the stream. | All |
| 1550 | Stream error. The header of the "start firmware download" is incorrect | General case: - Stream error: The header of the "start firmware download" is incorrect. | TCP/IP |
| 1551 | Stream error. A problem occurred when writing the downloaded program into flash memory | General case: - Stream error: A problem occurred when writing the downloaded program into flash memory. | TCP/IP |
| 1600 | TCP/IP error. Received a special TCP/IP record (record 0) with a node not equal to UltimET | General case: - TCP/IP error: Received a special TCP/IP record (record 0) with a node not equal to UltimET. | TCP/IP |
| 1601 | TCP/IP error. The TCP/IP reception buffer is full | General case: - TCP/IP error: The TCP/IP reception buffer is full. | TCP/IP |
| 1602 | TCP/IP error. The TCP/IP transmission buffer is full | General case: - TCP/IP error: The TCP/IP transmission buffer is full. | TCP/IP |
| 1603 | TCP/IP error. Keep alive timer has expired. Too long time without communication on an open TCP/IP port | General case: - TCP/IP error: Keep alive timer has expired. Too long time without communication on an open TCP/IP port. | TCP/IP |
| 1649 | TCP/IP error. The TCP/IP stack is in error | General case: - TCP/IP error: The TCP/IP stack is in error. | TCP/IP |
| 1650 | TCP/IP error. The transmitted message has to be segmented | General case: - TCP/IP error: One of the transmitted messages requires fragmentation. | TCP/IP |
| 1651 | TCP/IP error. One of the received messages was segmented | General case: - TCP/IP error: One of the received messages was segmented. | TCP/IP |
| 1652 | TCP/IP error. DHCP failed | General case: - TCP/IP error: DHCP failed. | TCP/IP |
| 1653 | TCP/IP error. The received message is bigger than the MTU | General case: - TCP/IP error: The received message is bigger than the MTU. | TCP/IP |
| 1654 | TCP/IP error. The IP checksum is incorrect | General case: - TCP/IP error: The IP checksum is incorrect. | TCP/IP |
| 1655 | TCP/IP error. One of the IP messages belongs to a protocol different than ICMP, TCP, and UDP | General case: - TCP/IP error: One of the IP messages belongs to a protocol different than ICMP, TCP, and UDP. | TCP/IP |
| 1656 | TCP/IP error. One of the transmitted messages requires fragmentation | General case: - TCP/IP error: One of the transmitted messages requires fragmentation. | TCP/IP |
| 1657 | TCP/IP error. Port not reachable | General case: - TCP/IP error: Port not reachable. | TCP/IP |
| 1658 | TCP/IP error. Unrecognized ICMP message | General case: - TCP/IP error: Unrecognized ICMP message. | TCP/IP |
| 1659 | TCP/IP error. The DHCP server has assigned an IP address already in use | General case: - TCP/IP error: The DHCP server has assigned an IP address already in use. | TCP/IP |
| 1699 | TCP/IP error. The TCP/IP stack has issued a warning | General case: - TCP/IP error: The TCP/IP stack has issued a warning. | TCP/IP |
| 1800 | External IO Error: Attempts to open the TCP/IP communication with external IO modules have failed 5 times in a row | General case: - External IO Error: Attempts to open the TCP/IP communication with external IO modules have failed 5 times in a row. | All |
| 1810 | External IO Error: The Ethernet link is broken (e.g. no cable) | General case: - External IO Error: The Ethernet link is broken (e.g. no cable). | All |
| 1815 | External IO error: waiting for the client Ethernet link has timed-out | General case: - External IO Error: Waiting for the client Ethernet link has timed-out. | All |
| 1840 | External IO Error: The client communication has been closed by the peer server | General case: - External IO Error: The client communication has been closed by the peer server. | All |
| 1850 | External IO Error: This UltimET board does not support TCP/IP external IOs | General case: - External IO Error: This UltimET board does not support TCP/IP external IOs. | All |
| 1920 | External IO Error: A Modbus exception has occurred | General case: - External IO Error: A Modbus exception has occurred. | All |
| 1970 | External IO Error: Access the external IO modules has timed out | General case: - External IO Error: Access the external IO modules has timed out. | All |
| 1990 | External IO error: no external input and no external output | General case: - External IO Error: No external input and no external output. | All |
| 2000 | Interpolation error. The number of parameters is incorrect | General case: - Interpolation error: The number of parameters is incorrect. | All |
| 2001 | Interpolation error. Incorrect parameter format | General case: - Interpolation error: Incorrect parameter format. | All |
| 2002 | Interpolation error. ISET command. Node resolution missing | General case: - Interpolation error: ISET command. Check if M239 is not equal to 0 for some nodes. | All |
| 2003 | Interpolation error. ISET command. One node is present in the definition of several axes. | General case: - Interpolation error: Iset command. One node is present in the definition of several axes. | All |

| *M64 | Description | Solutions | Product |
|------|-------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|---------|
| 2004 | Interpolation error. ISET command. The parameters are different for controllers belonging to the same axis. | General case: - Interpolation error: Iset command. The parameters are different for drives belonging to the same axis. | All |
| 2005 | Interpolation error. IBEGIN command. No node is defined. | General case: - Interpolation error: Ibegin command. No node is defined. | All |
| 2006 | Interpolation error. IBEGIN command. One node is not present. | General case: - Interpolation error: Ibegin command. One node is not present. | All |
| 2007 | Interpolation error. IBEGIN command. Error during the request of node's parameters. | General case: - Interpolation error: Ibegin command. Error during the request of node's parameters. | All |
| 2008 | Interpolation error. ISET command. One node is not present. | General case: - Interpolation error: Iset command. One node is not present. | All |
| 2009 | Interpolation error. IBEGIN command. One node position is out of range for interpolation reference frame | General case: - Interpolation error: Ibegin command. The position of an axis is out of range. | All |
| 2010 | Interpolation error. ISET command. Error during the request of node's parameters. | General case: - Interpolation error: Iset command. Error during the request of node's parameters. | All |
| 2011 | Interpolation error. One parameter is out of range. | General case: - Interpolation error: One parameter is out of range. | All |
| 2012 | Interpolation error. The interpolation mode is not active. | General case: - Interpolation error: The interpolation mode is not active. | All |
| 2014 | Interpolation error. Bad parameter in jmp mark action. | General case: - Interpolation error: Bad parameter in jmp mark action. | All |
| 2015 | Interpolation error. Unknown function number in jmp mark action. | General case: - Interpolation error: Unknown function number in jmp mark action. | All |
| 2016 | Interpolation error. The sequence thread is already running | General case: - Interpolation error: Sequence label overflow in jmp mark action. | All |
| 2019 | Interpolation error. The interpolation group number is out of range | General case: - Interpolation error: The interpolation group number is out of range. | All |
| 2020 | Interpolation error. Interpolation group already used | General case: - Interpolation error: Interpolation group already used. | All |
| 2021 | Interpolation error. IBEGIN command: no acknowledge of ITP command | General case: - Interpolation error: IBEGIN command: no acknowledge of ITP command. | All |
| 2022 | Interpolation error. IEND command: no acknowledge of ITP command | General case: - Interpolation error: IEND command: no acknowledge of ITP command. | All |
| 2023 | Interpolation error. End of interpolation fails. Move was still in progress | General case: - Interpolation error: End of interpolation fails. Move was still in progress. | All |
| 2024 | Interpolation error. Error when all groups do not have the same interrupt refresh rate | General case: - Two interpolated groups in the same interrupt. | All |
| 2025 | Interpolation error. Command counter underflow | General case: - Interpolation error: Command counter underflow. | All |
| 2030 | Interpolation error. The reference frame cannot be changed during a move | General case: - Interpolation error: The reference frame can't be changed during a move. | All |
| 2031 | Interpolation error. The command has been aborted by another command (HLx, END) | General case: - Interpolation error: The command has been aborted by another command (HLx, END). | All |
| 2049 | Interpolation error. Internal error | General case: - Interpolation error: Internal error. | All |
| 2050 | Interpolation error. No interpolation with this UltimET | General case: - Interpolation error: No interpolation with this UltimET. | All |
| 2051 | Interpolation error. Y variable was used with a depth different than 0 | General case: - Interpolation error: Y variable was used with a depth different than 0. | All |
| 2060 | Interpolation error: axis power off when interpolation active | General case: - Interpolation error: axis power off when when interpolation active. | All |
| 2070 | Interpolation error. The node plti does match with interpolation refresh rate. | General case: - Interpolation error: The node plti does match with interpolation refresh rate. | All |
| 2071 | Interpolation error. The interpolation update rate is not an integer. | General case: - Interpolation error: The interpolation update rate is not an integer. | All |
| 3000 | Command error. Command's parameters not correct | General case: - Command error: Command's parameters not correct. | All |
| 3001 | Command error. The label number is not correct | General case: - Command error: The label number is not correct. | All |
| 3002 | Internal error: Refer to sub-Error code | General case: - Command error: Internal management error. | All |
| 3003 | Forward command cannot be processed | General case: - The command has been aborted (HLT, HLO...). - A node has disappeared on TransnET or TransnET is down. | All |
| 3004 | This command cannot be executed when a sequence is running | General case: - This command cannot be executed when a sequence is running. | All |

| *M64 | Description | Solutions | Product |
|------|------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| 3005 | Trace configuration error | General case: - Bad configuration of Traces registers. | All |
| 3006 | Command not allowed | General case: - Compiled Sequence Error: On a TCPIP UltimET, you cannot choose to place the compiled sequences stacks in internal memory. | All |
| 3500 | Axis in mask for synchronized homing not found | General case: - Synchronized homing: Node timeout. | All |
| 3501 | Inconsistent axes homing modes for synchronized homing | General case: - Synchronized homing: The homing type (K40 parameter) is not the same on all the nodes. | All |
| 3502 | Index synchronization. Internal error | General case: - Synchronized homing: Internal error. | All |
| 3503 | Homing mode not supported for synchronized homing | General case: - Synchronized homing: The current homing type (K40 parameter) is not supported. | All |
| 3504 | Index synchronization. A node is in error | General case: - Synchronized homing: A node is in error. | All |
| 3505 | Synchronized homing: no index found in either direction | General case: - Synchronized homing: Reference mark has not been detected. | All |
| 3506 | Index synchronization. A node is powered off | General case: - Synchronized homing: A node is power off. | All |
| 3507 | Synchronized homing: There still is an axis on limit switch despite moving out | General case: - Index synchronization: Error in homing process when leaving home switch or limit switch. Checks K48 parameter or DIN 2, 9 or 10. This error occurs when after the trip from K48 the home switch or limit switch is present. | All |
| 3508 | Index synchronization: command aborted | General case: - Index synchronization: The command has been aborted. This can happen if the Transnet falls or the application crashes during the synchronized homing process. | All |
| 3600 | RSH command: some axes remain in error despite having been reset | General case: - RSH command: Some axes remain in error despite having been reset | All |
| 3610 | Transnet communication error: missing axes at Transnet restart | General case: - Transnet communication Error: List of axes after the RSH is not the same as before the command was issued. | All |
| 3620 | Transnet communication error: time-out waiting for transnet to disappear after RSH | General case: - Transnet communication Error: Time-out waiting for transnet to disappear after RSH. | All |
| 3630 | Transnet communication error: time-out waiting for transnet to re-appear after RSH | General case: - Transnet communication Error: Time-out waiting for transnet to re-appear after RSH. | All |
| 3640 | Transnet startup error: Inconsistent number of controllers. | General case: - Check if there are two controllers configured with the same DIP switch configuration | All |
| 3800 | External IO Error: Invalid IO number | General case: - External IO Error: Invalid IO number. | All |
| 3810 | Local IO Error: Invalid IO number | General case: - Local IO Error: Invalid IO number. | All |
| 3910 | External IO Error: The number of analog outputs is greater than supported | General case: - External IO Error: The number of analog outputs is greater than supported. | All |
| 3920 | External IO Error: The number of analog inputs is greater than supported | General case: - External IO Error: The number of analog inputs is greater than supported. | All |
| 3930 | External IO Error: The number of digital outputs is greater than supported | General case: - External IO Error: The number of digital outputs is greater than supported. | All |
| 3940 | External IO Error: The number of digital inputs is greater than supported | General case: - External IO Error: The number of digital inputs is greater than supported. | All |
| 3950 | MODBUS time-out | General case: - MODBUS Timeout. | All |
| 4000 | FPGA version is not the expected one | General case: - The current FPGA is not the expected one. Please reboot your PC. | All |
| 4100 | Management error. Flash bad checksum | General case: - Management error: Flash bad checksum. | All |
| 4101 | Management error. Flash. Problem during the erasure of the flash | General case: - Management error: Flash. Problem during the erasure of the flash. | All |
| 4102 | Management error. Flash. Problem during the writing of the flash | General case: - Management error: Flash. Problem during the writing of the flash. | All |
| 4103 | Management error. Flash. Problem during the writing and check of the flash | General case: - Management error: Flash. Problem during the writing and check of the flash. | All |
| 4107 | Management error. Flash. A block is wrongly locked or unlocked | General case: - Management error: Flash. A block is wrongly locked or unlocked. | All |

| *M64 | Description | Solutions | Product |
|------|-----------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|--------------|
| 4108 | Management error. Flash. Incorrect sequence format | General case: - Management error: Flash has incorrect sequence format. | All |
| 4109 | Management error. Flash. This block is empty | General case: - Management error: Flash. This block is empty. | All |
| 4120 | HP Task duration > 100 us (Time us) | General case: - Management error: An error occurs due to a time overflow of the high priority interrupt. | All |
| 4200 | Management error. An error has been detected on a node present on the TransnET | General case: - Management error: An error has been detected on a node present on the TransnET. | All |
| 4210 | The PCI fpga found a problem of watchdog (too long time to acknowledge a PCI interrupt) | General case: - The PCI fpga found a problem of watchdog (too long time to acknowledge an PCI interrupt). | PCI, PCIe |
| 4500 | Invalid IO configuration | General case: - I/O error: Bad digital I/O setting interrupt. | All |
| 4501 | Error state triggered by a DIN | General case: - I/O error: The DIN are in the error state defined by K305. | All |
| 5800 | The distance between the two monitored registers has fallen below the specified limit | General case: - Check parameters of command CHKDISTGRT | All |

13. Service and support

For any inquiry regarding technical, commercial and service information relating to ETEL S.A. products, please contact your ETEL S.A. representative listed on our website (www.etel.ch).

The technical hotline, based in ETEL S.A.'s headquarters, can be reached by:

- Phone: +41 (0)32 862 01 12.
- Fax: +41 (0)32 862 01 01.
- E-mail: support@etel.ch.

Please refer to your corresponding ETEL S.A. representative for more information about the technical documentation. ETEL S.A. organizes training courses for customers on request, including theoretical presentations of our products and practical demonstrations at our facilities.

Index

A

Alias, 41
ASR, 82
ASW, 81

B

Block statement, 152
Break statement, 155

C

C common parameters
 C245, 138, 166
CH_BIT_REG32, 78
CHKDISTGRT, 45
CLRWAIT, 56
CLX, 44
Commands
 ASR, 82
 ASW, 81
 CH_BIT_REG32, 78
 CHKDISTGRT, 45
 CLRWAIT, 56
 CLX, 44
 DIN, 57
 DOUT, 58
 EIOMAXURST, 67
 EIOSTA, 63
 END, 164
 ERR, 74, 165
 FGA, 159
 FWD, 85
 GETDINS, 160
 GETDOUT, 160
 GETEAINCS, 161
 GETEAINRS, 161
 GETEAOUTC, 161
 GETEAOUTCS, 161
 GETEAOUTR, 161
 GETEAOUTRS, 161
 GETEDINS, 160
 GETEDOUT, 161
 GETEDOUTS, 161
 GETEREG, 161
 GETMDINS, 160
 GETMDOUT, 160
 GETMEDINS, 160
 GETMEDOUT, 161
 GETMEDOUTS, 161
 GGA, 159
 HLB, 124
 HLO, 124
 HLT, 124
 IABSCORDS, 95
 IABSMODE, 93
 IBEGIN, 91
 IBRK, 122
 ICAM, 125

ICCW, 99
ICCWR, 101
ICLRB, 123
ICONC, 107
ICONT, 123
ICW, 99
ICWR, 101
IEND, 97
ILINE, 99
ILOCK, 123
IMARK, 128
IMRES, 135
IMROT, 133
IMSCALE, 132
IMSHEAR, 134
IMTRANS, 131
INCONC, 107
IND, 49
IPT, 115
IPVT, 110
IPVTUPDATE, 113
ISET, 90
ISPDRATE
 in G-code, 105
 in PT, 115
 in PVT, 111
ISTP, 122
ITACC, 103
ITANSPDMASK, 127
ITDEC, 103
ITJRT, 104
ITSPD, 103
IULINE, 119
IUNLOCK, 123
IUNOCONC, 122
IURELATIVE, 121
IUSPDMASK, 120
IUSPEED, 119
IUTIME, 121
IWTT, 124
JMP, 162
NEW, 47
RCT, 68
RES, 47
RIC, 67
RST, 76
RSTDOUT, 160
RSTEDOUT, 160
RSU, 48
SAV, 46
SER, 49
SET_RANGE, 45
SETDOUT, 160
SETEAOUTC, 161
SETEAOUTR, 161
SETEDOUT, 160
SETEREG, 161
SETMDOUT, 160
SETMEDOUT, 161
SETRTV, 84
SSR, 78

VER, 49
 WBC, 55
 WBS, 55
 WPG, 56
 WPL, 56
 WSBC, 55
 WSBS, 55
 WTD, 73
 WTK, 56, 129
 WTM, 54
 WTT, 54
 ZTE, 70
 Commands list, 168
 Commands management, 17
 Commands syntax, 40
 Commands types, 15
 Concatenation, 106
 Continue statement, 156
 Conversion factor, 135

D

Digital inputs/outputs, 57
 DIN, 57
 Distance limits, 89
 Do statement, 154
 DOUT, 58

E

EIOMAXURST, 67
 EIOSTA, 63
 Electrical interface, 31
 END, 164
 ERR, 74, 165
 Errors list, 193
 Errors management, 73, 164

F

FGA, 159
 For statement, 154
 FWD, 85

G

G-code mode, 98
 GETDINS, 160
 GETDOUT, 160
 GETEAINCS, 161
 GETEAINRS, 161
 GETEAOUTC, 161
 GETEAOUTCS, 161
 GETEAOUTR, 161
 GETEAOUTRS, 161
 GETEDINS, 160
 GETEDOUT, 161
 GETEDOUTS, 161
 GETEREG, 161
 GETMDINS, 160
 GETMDOUT, 160
 GETMEDINS, 160
 GETMEDOUT, 161
 GETMEDOUTS, 161

GGA, 159

H

HLB, 124
 HLO, 124
 HLT, 124
 Homing, 49

I

IABSCORDS, 95
 IABSMODE, 93
 IBEGIN, 91
 IBRK, 122
 ICAM, 125
 ICCW, 99
 ICCWR, 101
 ICLRB, 123
 ICONC, 107
 ICONT, 123
 ICW, 99
 ICWR, 101
 IEND, 97
 If statement, 152
 ILINE, 99
 ILOCK, 123
 IMARK, 128
 IMRES, 135
 IMROT, 133
 IMSCALE, 132
 IMSHEAR, 134
 IMTRANS, 131
 INCONC, 107
 IND, 49
 Inputs, 57, 160
 Installation, 24
 Internal functioning, 15
 Interpolation, 88
 movement, 98
 IPT, 115
 IPVT, 110
 IPVUPDATE, 113
 ISET, 90
 ISPD RATE
 in G-code, 105
 in PT, 115
 in PVT, 111
 ISTOP, 122
 ITACC, 103
 ITANSPDMASK, 127
 ITDEC, 103
 ITJRT, 104
 ITSPD, 103
 IULINE, 119
 IUNLOCK, 123
 IUNOCONC, 122
 IURELATIVE, 121
 IUSPDMASK, 120
 IUSPEED, 119
 IUTIME, 121
 IWTT, 124

J

JMP, 162

K

K parameters

K32, 81, 82

K33, 58

K120, 68

K121, 68

K122, 68

K123, 68

K124, 68

K125, 69

K126, 69

K127, 69

KD127, 69

KF127, 69

KL127, 69

K128, 69

K129, 69

K140, 75

K141, 74

K142, 164

K143, 142

K144, 164

K166, 76

K171, 58

K177, 77

K198, 44

K200, 166

K297, 138

K302, 166

K305, 58

K353, 59

K357, 59

K360, 59

K415, 166

K513, 18

K522, 88

K523, 88

K524, 88

K525, 88

K526, 88

K530

in G-code, 105

in PT, 115

in PVT, 111

K532, 123

K533, 123

K540, 128, 129

K541, 129

K545, 120

K546, 111, 115

K560, 113

K561

in PT, 116

in PVT, 111

K580, 135

K581, 135

K582, 135

K595, 28, 30, 57

K600, 97

K610, 66

K611, 66

K615, 65

K616, 65

K620, 66

KF620, 66

K621, 66

KF621, 66

K625, 65

KF625, 65

K626, 65

KF626, 65

KL630, 74

K631, 73

KL632, 73

K717, 125

K721, 60

K722, 60

K724, 60

K725, 61

K726, 61

K727, 61

K729, 61

K734, 61

K740, 65

KF740, 65

K741, 65

KF741, 65

K760, 64

L

LEDs meaning, 28, 30, 38

M

M monitorings

M50, 57

M63, 77

M64, 193

M66, 76

M70, 48

M71, 48

M72, 49

M73, 49

M85, 49

M87, 15

M97, 77, 140

M101, 54

M120, 69

M121, 69

M122, 69

M123, 69

M124, 69

M125, 69

M126, 69

M127, 69

MF127, 69

ML127, 69

M128, 69

M129, 69
 M131, 69
 M134, 69
 ML134, 69
 M135, 69
 M136, 69
 M141, 74
 M166, 76
 M171, 58
 M201, 166
 M202, 166
 M203, 60
 M228, 71
 ML228, 71
 M282, 48
 M417, 166
 M418, 166
 M443, 82
 M447, 82
 M448, 82
 M449, 82
 M450 to M465, 83
 MD450 to MD457, 83
 MF450 to MF465, 83
 ML450 to ML457, 83
 M512, 18
 ML512, 15
 M513, 18
 M516, 159
 M518, 128
 M519, 128
 M520, 77, 82
 M524, 81
 M525, 81
 ML525, 91
 M526, 81
 M530, 92
 M531, 124
 M535, 19
 M590, 94
 M629, 75
 M640, 98
 M641, 98
 M642, 98
 M643, 98
 M644, 98
 M645, 98
 M646, 98
 M647, 98
 M649, 103
 M650, 103
 M651, 103
 M652, 104
 M658, 89
 M659, 89
 M660, 98
 M661, 98
 M662, 98
 M663, 98
 M664, 98
 M665, 98

M666, 98
 M667, 98
 M669, 103
 M670, 103
 M671, 103
 M672, 104
 M680, 127
 MF680, 127
 M681, 127
 M690, 62
 M691, 62
 M692, 62
 M693, 62
 M694, 62
 M695, 62
 M696, 67
 M697, 67
 M698, 61
 M699, 61
 M705, 61
 M706, 67
 M707, 60, 61
 M708, 67
 M709, 67
 M712, 67
 M726, 91
 M727, 91
 M728, 91
 M729, 91
 M730, 91
 M731, 91
 M732, 91
 M733, 91
 M740, 66
 MF740, 66
 M741, 66
 MF741, 66
 M745, 64, 66
 MF745, 64
 MF746, 64
 M760, 64
 M764, 63
 M765, 64
 Marks, 128
 Matrix, 131
 Monitorings list, 187
 Moving bit, 78

N

NEW, 47

O

On-fly trajectory change, 112
 Ordering, 22
 Ordering information, 22
 Outputs, 57, 160

P

Parameters list, 183
 Predefined functions, 157

Programming, 138
 basic, 138
 structured code, 143
PT mode, 114
PVT mode, 108

R

RCT, 68
Real-time channel, 79
Record of revisions, 9
Reference frame, 92
Registers
 K parameters, 41
 M monitoring, 41
 maximum value, 44
 minimum value, 44
 value attribution, 42
 X user variables, 41
RES, 47
Reset, 48
Reset bit management, 78
Return statement, 156
RIC, 67
Rotation, 133
RST, 76
RSTDOUT, 160
RSTEDOUT, 160
RSU, 48
RTV, 79

S

Safety signals on DOUT, 59
SAV, 46
Scaling, 132
Sequence gate, 158
SER, 49
Set bit management, 78
Set several registers, 78
SET_RANGE, 45
SETDOUT, 160
SETEAOUTC, 161
SETEAOUTR, 161
SETEDOUT, 160
SETEREG, 161
SETMDOUT, 160
SETMEDOUT, 161
SETRTV, 84
Shearing, 134
Software installation (DSMAX3), 26, 29
SSR, 78
Statements, 151
Status, 77
Switch statement, 153
Synchronized homing, 49

T

Tangential speed, 127
Thread, 21, 140, 163
Traces management, 68
Translation, 131

U

ufai, 88
ufpi, 88
ufsi, 88
ufti, 88
ULM movement, 116
Unit conversion, 156
Units, 88

V

Variables X, 141
VER, 49

W

Wait commands, 54
Watchdog setting, 161
WBC, 55
WBS, 55
While statement, 154
WPG, 56
WPL, 56
WSBC, 55
WSBS, 55
WTD, 73
WTK, 56, 129
WTM, 54
WTT, 54

Z

ZTE, 70