ULTIMET ADVANCED Motion Controller

Operation & Software Manual

Version E

ETEL

THIS PAGE IS INTENTIONALLY LEFT BLANK

Table of contents

Chapter A: Internal operation & generalities

1 Overview 16
1.1 Acronyms 16
2 Internal operation 17
2.1 Communication with the ULTIMET ADVANCED 17
2.2 Commands 17
2.3 Block diagram 18
2.3.1 Communication 18
2.3.2 Command sources 18
2.4 Customization 19

Chapter B: Installation and setting

3 Installation 22
3.1 TCP/IP connections 22
3.1.1 User application 22
3.1.2 Commissioning 22
3.2 PCIe connection 23
3.2.1 PCIe driver installation 23
3.2.2 ULTIMET ADVANCED boot up 23
4 Configuration 27
4.1 TCP/IP connection 27
4.2 PCIe connection 29
4.3 Firmware download 30
4.4 ULTIMET ADVANCED reset 31

Chapter C: Functionalities

5 General commands & registers syntax 34
5.1 Commands 34
5.2 Registers 35
5.2.1 Registers group 35
5.2.2 Register specification (ComET terminal) 36
5.2.3 Register assignment (ComET terminal) 37
6 Registers handling 39
6.1 CLX command 39

6.2 SET_RANGE command 39
6.3 Set / reset bit(s) management 39
6.4 Set several registers 40
6.5 CHKDISTGRT command 40
6.6 Saving and restoring the registers 41
7 WAIT Commands 43
7.1 WTT command 43
7.2 WTM commands 43
7.3 Wait on bits: WBS, WBC, WSBS and WSBC commands 44
7.4 Wait on values: WPL and WPG commands 45
7.5 Clearing a wait command 45
8 Status 46
8.1 ULTIMET ADVANCED controller status 46
8.2 User status parameter 46
8.3 Temperature 46
8.4 Errors and warnings management 47
8.4.1 Error notification 47
8.4.2 Reaction to errors 47
8.4.3 Errors propagation on TransnET 47
8.4.4 TransnET communication errors 48
8.4.5 The *ERR command 49
8.4.6 Resetting errors 49
8.4.7 Warnings 50
8.5 User LEDs 50
8.6 Firmware logs 51
9 Synchronized homing on several axes 52
10 Traces Management 57
10.1 Traces configuration 57
10.1.1 Pre- and post-trigger capabilities 58
10.2 Non-synchronized mode 59
10.3 Synchronized mode 60
10.4 Trigger condition test and edge detection 61
10.5 Traces upload 61
10.6 Continuous traces 61
11 ULTIMET ADVANCED software characteristics registers 63
12 ETEL Software Module 64

12.1 Common registers 64
12.2 RTV consideration 66
12.2.1 Commands 66
12.3 ESM error codes 67
13 Local digital Inputs / Outputs 68
13.1 Digital inputs 68
13.1.1 Direct digital input 68
13.1.2 Protection signals on digital inputs 70
13.1.3 Troubleshooting 70
13.2 Digital outputs 71
13.2.1 Direct digital output 71
13.2.2 Protection signals on digital output 73
13.2.3 TransnET signal output 74
13.2.4 Interpolation digital output 74
13.2.5 Troubleshooting 75
14 Real-time channels on TransnET 76
14.1 Principle 76
14.2 Data sharing 76
14.2.1 Data sharing between controllers 76
14.2.2 Data sharing between controllers and a PC application (PCIe only) or a CSM 77
14.2.3 Disappearance of TransnET 77
14.3 Slot management functions 78
14.4 UltimET – AccurET data sharing functions 79
14.4.1 ASW (ASsign for Write) command 79
14.4.2 ASR (ASsign for Read) command 80
14.4.3 SETRTV command 82
14.5 PC (through PCIe) or CSM application data sharing functions 84
14.5.1 Introduction 84
14.5.2 Access to RTV slots from the application 84
14.5.3 Cyclic RTV task 84
14.6 Real-time channels timing 87
15 Interpolation 88
15.1 Introduction 88
15.1.1 Management of the interpolation commands 88
15.1.2 Management of a mix of normal and interpolated commands 90
15.1.3 Units of the interpolation commands 92
15.1.4 Distance and time limits for interpolation commands 93
15.2 Setting of the interpolation 94
15.2.1 ISET command 94

15.2.2 IBEGIN command 96
15.2.3 IEND command 101
15.2.4 Two groups of interpolation 101
15.2.5 EDI related functions 102
15.3 User position vs. real position 103
15.4 Interpolation movements 103
15.4.1 'G-code based' mode 103
15.4.2 PVT (Position-Velocity-Time) mode 114
15.4.3 PT (Position-Time) mode 119
15.4.4 Braking commands 121
15.4.5 Time commands 124
15.5 Status 128
15.6 Marks 129
15.6.1 Identification of the mark 129
15.6.2 Action associated with a mark 129
15.7 Matrix 131
15.7.1 Translation function 132
15.7.2 Scaling function 133
15.7.3 Rotation function 134
15.7.4 Shearing function 134
15.7.5 Matrix restore 135
15.8 User conversion factor 136
16 SPI feature 137
16.1 I/O configuration 137
16.2 SPI connection /disconnection 137
16.2.1 Connection 137
16.2.2 Disconnection 138
16.2.3 CRC handling 138
16.3 SPI read-write commands 139
16.3.1 Commands description 139
16.3.2 Abort command 140
16.4 Transmission status 141
16.5 Cyclic transfer mode 141
16.5.1 Feature 141
16.5.2 Amount of bytes that can be transferred 142
16.5.3 Watchdog and utility functions 142
16.5.4 Cyclic handler example 142
16.6 Burst transfer mode 144
17 Feature licenses 145
17.1 Management 145

17.2 Configuration 145
17.3 Example 146

Chapter D: Programming

18 Customer Software Module 150
18.1 Prerequisites 150
18.2 Toolchains 150
18.3 General principle 150
18.4 FreeRTOS introduction 150
18.5 Firmware – CSM communication 151
18.6 Recommendations 152
19 Development environment 154
19.1 Overview 154
19.2 CSM template package 155
19.2.1 Content 155
19.2.2 Installation 155
19.3 Working with the Xilinx SDK 156
19.3.1 Installation 156
19.3.2 Setting up 163
19.3.3 CSM version and description information 169
19.3.4 Compiling and linking 170
19.3.5 CSM template execution 173
19.4 CSM Template Visual Studio project 180
19.4.1 Installing 180
19.4.2 Building 181
19.4.3 Running 183
19.4.4 Renaming 186
20 CSM template 188
20.1 Structure 188
20.1.1 User interface (CSMUserIface.*) 188
20.1.2 System (CSMSystem.*) 192
20.1.3 Terminal (CSMTerminal.*) 196
20.1.4 Template (CSMTemplate.c) 196
20.2 Workflow 197
20.3 EDI update 198
21 CSM special features 199
21.1 High priority real-time RTV handler 199
21.2 Medium priority real-time Transfer handler 199

21.3 Getting the current UA UTC time from a CSM 200
22 CSM control from a host application 201
22.1 Start a CSM 201
22.2 Waiting for a CSM to be started 201
22.3 Run a CSM high-level function 202
22.4 Stop a CSM 202
22.5 Store a CSM on the permanent storage 202
23 CSM troubleshooting 204
23.1 Debug stdio/stdout 204
23.1.1 TCP/IP debug server 204
23.1.2 Opening a Telnet terminal in the Xilinx SDK 206
23.2 File logging 208
23.3 FTP access to User SD card 211
23.3.1 FTP from windows command shell 211
23.3.2 FTP Access through Windows Explorer 211
24 EDI functions not supported in a CSM 215
25 Special note about dynamic memory allocation 216
26 Special note about printf suite functions 217

Chapter E: Appendixes

27 ULTIMET ADVANCED – UltimET Light differences 220
27.1 New features 220
27.1.1 Customer Software Module (CSM) 220
27.1.2 Several application interfaces (TCP/IP – PCIe – CSM) 220
27.1.3 Configurable IOs 220
27.1.4 SPI (Serial Peripheral Interface) feature 220
27.1.5 More RTV slots accessible via ASR/ASW 220
27.1.6 SD cards 221
27.2 Features that behave differently 221
27.2.1 PCIe watchdog 221
27.2.2 Emergency halt features 221
27.3 Features not in ULTIMET ADVANCED 221
27.3.1 Interpolation Unified Linear Move (ULM) mode 221
27.3.2 Get / Free Gate command 222
27.3.3 Nodes status registers 222
28 Troubleshooting 223
28.1 Recovery firmware 223

28.2 Troubleshooting hints 223
28.2.1 JTAG connection not accessible 224
28.2.2 ULTIMET ADVANCED PCIe connection not accessible 224
28.2.3 CSM troubleshooting 224
28.2.4 SPI troubleshooting 224
29 Commands list 226
30 K parameters list 241
31 M monitoring registers list 245
32 Warnings reference list 254
33 Errors reference list 255
33.1 Firmware error codes 255
33.2 ESM error codes 264
34 Service and support 266

THIS PAGE IS INTENTIONNALLY LEFT BLANK

Record of revisions:

Document revisions		
Version	**Date**	**Main modifications**
Ver A	14.09.20	First version
Ver B	15.09.21	Updated version (with firmware 3.21A) - PCIe connection updated (refer to §3.2) - Development environment updated (refer to §19) - Waiting for a CSM to be started (refer to §15.4.4.4 and §22.2)
Ver C	11.03.22	Updated version (with firmware 3.22A) - CSM version and description information (refer to §19.3.3) - Getting the current UA UTC time from a CSM (refer to §21.3) - TCP/IP debug server updated (refer to §23.1.1) - Special note about printf suite functions (refer to §26) - Minor changes (refer to modification strokes)
Ver D	26.08.22	Updated version (with firmware 3.23A) - New command: RSU (refer to §4.4)
Ver E	13.03.23	Updated version (with firmware 3.50A) - ESM description updated (refer to §12) - EDI update (refer to §20.3) - Feature licenses (refer to §17)

Documentation concerning the ULTIMET ADVANCED:

• **ULTIMET ADVANCED Operation & Software Manual**	**ULTIMET ADVANCED setup, use & programming**
• ULTIMET ADVANCED Hardware Manual	Specifications and electrical interfaces
• AccurET and ACCURET+ Operation & Software Manual	Setup, use and programming
• AccurET Modular48 Hardware Manual	Specifications and electrical interfaces
• AccurET Modular300 Hardware Manual	Specifications and electrical interfaces
• AccurET Modular400-600 Hardware Manual	Specifications and electrical interfaces
• AccurET VHP48 Hardware Manual	Specifications and electrical interfaces
• AccurET VHP100 Hardware Manual	Specifications and electrical interfaces
• ACCURET+ 100 Hardware Manual	Specifications and electrical interfaces
• ACCURET+ 300 Hardware Manual	Specifications and electrical interfaces
• ComET User's Manual	Commissioning software
• QCT User's Manual	QUIET Commissioning Tool software
• EDI User's Manual	EDI principle and operation

The ULTIMET ADVANCED motion controller has been successfully tested and evaluated to meet the UL 61010-1 standard for the US Market and CAN/CSA-C22.2 No. 61010-1 for the Canadian market.

Introduction

ULTIMET ADVANCED is a multi-axis motion controller belonging to the UltimET family and it is intended to work with the AccurET and ACCURET+ position controllers.
The purpose of this manual is to provide detailed information concerning the setup, operation and programming of the ULTIMET ADVANCED (also referred to as “motion controller” in this document).
Detailed information regarding the electrical and mechanical specifications, installation and interfacing of the ULTIMET ADVANCED is provided in the corresponding ‘Hardware Manual’.

The information provided in this manual is valid for ULTIMET ADVANCED 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: Changes between two successive versions of this manual are highlighted with a modification stroke in the margin.

Safety

	The user must have read and understood this documentation as well as those listed in page 11 before carrying out any operation on this product. Please contact ETEL S.A. or authorized distributors in case of missing information or question regarding the installation procedures, safety or any other issue.
	ETEL S.A. disclaims all responsibility for accidents and damages if the safety instructions, the procedures and the usage described in the present manual and in those listed in page 11 are not followed.

- Never use the motion controller for purposes other than those described in this manual.
- A competent and trained technician must install and operate the motion controller, in accordance with all specific regulations of the respective country concerning both safety and EMC aspects.
- Troubleshooting and servicing are permitted only by ETEL's technicians and agreed distributors.
- The user must provide at anytime the appropriate protections against direct contact with electrical and moving parts of the connected system. Operating the motion controller may make the motor move.
- The safety symbols placed on the motion controller or written in the manuals listed in page 11 must be respected.
- If the motion controller 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 controller.

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

General operating conditions

Refer to the corresponding “Hardware Manual” for information related to the operating conditions of this motion controller.

Structure of this manual

This user’s manual consists of sections:
- Chapter A: Internal operation and generalities.
- Chapter B: Installation and setting.
- Chapter C: Advanced commands.
- Chapter D: Programming.
- Chapter E: Appendixes.

If you are not an experienced user, read first chapter A (page 15) to understand the basics about the operation of the ULTIMET ADVANCED motion controller.

Then, refer to chapter B (page 21) to successfully realize a first installation and setup of the motion controller.

Chapter C (page 33) covers in detail the different functionalities supported by the ULTIMET ADVANCED, while Chapter D (page 149) describes how to program it, including instructions on how to install and configure the development environment.

The appendixes in chapter E (page 219) include a summary of the differences between the UltimET Light and ULTIMET ADVANCED motion controllers and provide a few troubleshooting hints. It also covers the registers and commands lists, as well as error and warning messages lists.

Remark: ETEL can provide its customers with training courses including theoretical presentations and practice in real conditions, at our facilities in Môtiers (Switzerland).

THIS PAGE IS INTENTIONALLY LEFT BLANK

Chapter A: Internal operation & generalities

1 Overview

The ULTIMET ADVANCED is a powerful and versatile multi-axis motion controller belonging to the UltimET family. On one side, this motion controller can interface an application/commissioning host PC using a cabled PCIe connection or one of the three TCP/IP connections available. On the other, the ULTIMET ADVANCED is intended to control position controllers of the AccurET family connected via the gigabit real-time proprietary TransnET communication bus. For further information on the available interfaces, please refer to the corresponding 'Hardware Manual'.
The ULTIMET ADVANCED supports by default the interpolation mode, enabling the execution of interpolated trajectories on 2 groups of up to 4 interpolation axes (X, Y, Z and θ directions) with a very fast cycle time (100 µs). Each interpolation axis may represent several position controller axes (e.g. this feature can be used with a gantry system where 2 axes are controlled by the same reference). The trajectories available can be used with a simple language based on G-Code or complete PVT or PT modes with points refreshed every 100 µs. The ULTIMET ADVANCED includes eight software programmable digital I/Os. Any four of these IOs can also be configured to operate as one standard Serial Peripheral Interface (SPI), facilitating the integration with third-party equipment.

The user-friendly ComET software can be used to configure and monitor the operation of the ULTIMET ADVANCED and AccurET controllers on the TransnET bus (for further information refer to the 'ComET User's Manual').

A software library EDI (...) is provided allowing the user to write its own PC-based program in C to control the ULTIMET ADVANCED and AccurET controllers. An embedded version of this library can be used to write a user application running on the ULTIMET ADVANCED, referred to as the Customer Software Module (CSM). For further information about the CSM and programming capabilities of this product, please refer to chapter D (page 149).

In addition, the ULTIMET ADVANCED can host customization plug-ins that implement enhancements as required by specific applications. These are proprietary application oriented extensions of the firmware provided by ETEL and are referred to as ETEL Software Modules (ESM). For further information about the ESM refer to §12.

The ULTIMET ADVANCED features two SD cards. One is a standard SD card available exclusively for the user's embedded application data, while the other is a micro-SD card reserved for internal data storage (e.g. internal logging and register files), CSM binary files and firmware.

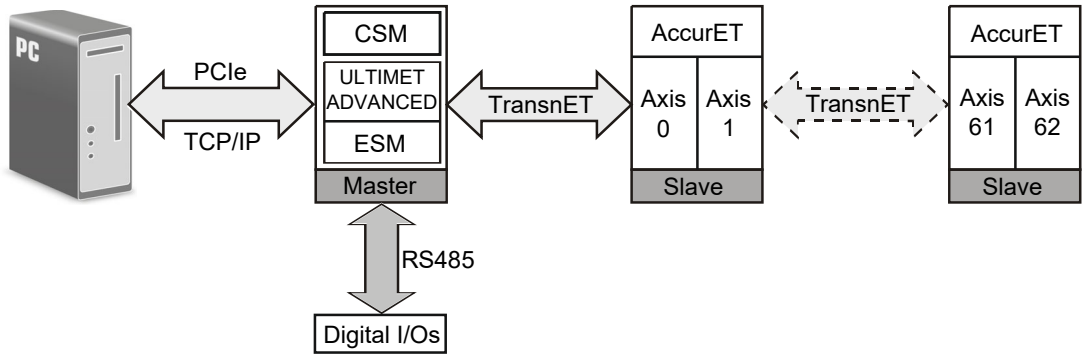
1.1 Acronyms

Acronyms	Definition
CSM	Customer Software Module
EDI	ETEL Device Interface
EMC	Electromagnetic Compatibility
EOF	End of file
ESD	Electrostatic Discharge
ESM	ETEL Software Module
IDE	Integrated Development Environment
I/O	Input / Output
MMF	Moving Metrology Frame
OS	Operating System
PC	Personal Computer
PCIe	Peripheral Component Interconnect express
PT	Position – Time
PVT	Position – Velocity – Time
RTOS	Real-Time Operating System
SD	Secure Digital
SDK	Software Development Kit
SMP	Single Symmetric Multiprocessing
SPI	Serial Peripheral Interface
RTV	Real-Time Value
TCP/IP	Transmission Control Protocol / Internet Protocol
UA	ULTIMET ADVANCED

2 Internal operation

2.1 Communication with the ULTIMET ADVANCED

The PC(s) and ULTIMET ADVANCED are linked through PCIe or TCP/IP (or both). All AccurET position controllers are linked together to the ULTIMET ADVANCED (master) by the TransnET communication bus. One of the roles of the master is to dispatch commands received from the host PC or the embedded user application (CSM) to the position controllers and relay back the execution status. Each axis has a unique number and, if several axes are chained, their numbers must be different from each other (from 0 to 62). The ULTIMET ADVANCED will always have the number 63. It is thus possible to link up to 63 axes.



The monitoring register ***ML512** allows the user to know the number of axes present on the TransnET bus. Refer to §5.2 for more information about the monitoring registers.

Monitoring	Comment
***ML512**	Gives the mask of the axes present on the TransnET
***M87**	Gives the ULTIMET ADVANCED axis number

2.2 Commands

The ULTIMET ADVANCED supports different types of commands:

- Normal: general-purpose commands to manage the axes, control the CSM's execution, perform mathematical operations (e.g. addition, substraction, multiplication, division), etc.
- Interpolation: commands to describe the interpolated movements to be executed by the axes (circles, lines, etc.).
- Urgent: commands to execute a quick action on an axis. There are specific channels reserved for these commands, so they are processed before any other type of command. These are mainly protection related commands, such as immediate stop, stop with programmed deceleration, power off, etc. Urgent commands can be sent from the 'Terminal' tool of ComET, from the resident program running on the host PC or even from the embedded application running on the ULTIMET ADVANCED (CSM).
- Monitoring requests enable the user to obtain information about an axis (position, speed, parameter value, etc.) and these are processed as urgent commands.
- Local stream command: used to transfer files and traces acquisition data.
- ESM commands: these commands allow the user to configure and control the specific feature implemented by the ESM. They are either processed as normal or urgent commands depending on the intended action. There are also ESM specific registers, which are only available when there is an ESM loaded.

To allow the different application interfaces to send commands over TransnET without blocking each other, the TransnET contains 4 gates. 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.

With the ULTIMET ADVANCED, the gates are used as follow:
- Gate 0: TCPIP connections
- Gate 1: PCIe
- Gate 2: CSM
- Gate 3: Reserved

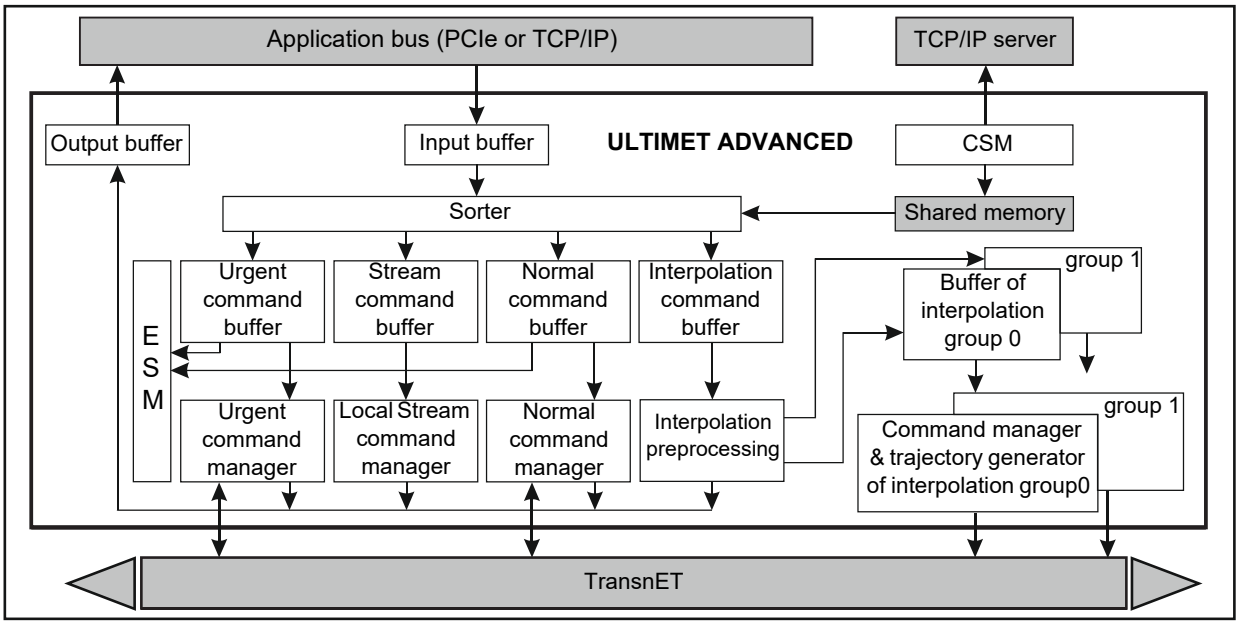
Warning: The gates allow the transfer of normal commands without blocking each other, however it is not possible to perform everything in parallel. E.g. it is neither possible to execute simultaneous upload of Trace acquisitions, nor download/upload of sequences, nor download/upload of the firmware. In addition, it is not possible to simultaneously upload/download registers with the Stream feature.

2.3 Block diagram

2.3.1 Communication

The ULTIMET ADVANCED includes six communication ports:
- One PCIe and two TCP/IP ports
  These ports allow the motion controller to communicate with one or several host PCs. Through these ports, the user's application can send commands to and receive information from the ULTIMET ADVANCED.
- A shared memory port
  This is an internal port for the communication between the embedded user application (CSM) and the ULTIMET ADVANCED. ETEL provides a C function library (EDI) running on FreeRTOS for using this interface.
- A TCP/IP port dedicated to the CSM
  This port can be used at the user's own discretion, like e.g. as a dedicated channel to convey information directly out from the CSM to a host PC or for debugging purposes.

A TransnET port to exchange information in real-time with AccurET position controllers. A direct TransnET access for RTV communication is allowed through either PCIe or CSM.



2.3.2 Command sources

The commands sent to the different axes (ULTIMET ADVANCED included) can be issued from different sources. All sources can be active without disrupting each other. The sources can be either the embedded user application (CSM) running in the ULTIMET ADVANCED or the user application running on a host PC.

The transmission and execution of a "normal" command are done by the normal command manager (refer to the diagram above). Commands are processed in the order of arrival. If a command cannot be immediately

executed (for e.g. if an axis is already busy), the manager simply goes to the next one and returns later to this command. Normal commands may take several TransnET bus cycles to be executed or, for local commands, several firmware execution cycles.

Urgent commands sent over TransnET are executed in just one bus cycle and local urgent commands are completely executed in one firmware execution cycle.

The local stream manager is used to transfer data packets when downloading a firmware, or uploading traces acquisitions. There can only be one data stream running at a given time, but it can be simultaneous with the processing of other commands types.

The interpolation commands are all pre-processed, placed in the interpolation command buffer (one per interpolation group) and executed by their interpolation command manager and trajectory calculator in the order of arrival. For further information about the interpolation functionality refer to §15

As the TransnET bandwidth is not infinite, it can, therefore, be “fully occupied” at a given moment. In such a situation, commands pile up in the different buffers up to the saturation of the input buffer, in which case an alert is issued to the connected applications (on a host PC or an embedded CSM). Dispatching of commands is resumed as soon as the TransnET has available bandwidth.

2.4 Customization

The ULTIMET ADVANCED is a versatile multi-axis controller that can be customized in two different ways:

- By the user, by writing his own program to be executed directly on the ULTIMET ADVANCED. This is named a Customer Software Module (CSM) and can be considered the equivalent of the “sequences” running on the UltimET Light and AccurET controllers, but much more powerful and with less limitations. With the ULTIMET ADVANCED, the user can design his application to execute the real-time portion directly on the motion controller as a CSM and leave the non-real time processing to run on the host PC. The CSM is written as a C program, compiled using Xilinx SDK environment and linked to EDI libraries ported to the FreeRTOS operating system. A core of the ULTIMET ADVANCED Quad-Core ARM processor is dedicated to the execution of the CSM. For further details on the capabilities and limitations of the CSM, please refer to chapter D (page 149).
- By ETEL, by writing customization plug-ins adding functionality to enhance the performance of ETEL’s motion systems. Such plug-ins are named ETEL Software Modules (ESM). For further information about ESM refer to §12.

THIS PAGE IS INTENTIONALLY LEFT BLANK

Chapter B: Installation and setting

3. Installation

The ULTIMET ADVANCED can accept simultaneously several TCP/IP connections and one PCIe connection.

Three different connectors are available for TCP/IP communication, each accessible using three different IP addresses:

- Connector X1 (TCP/IP communication #1): interface for the user application or commissioning purposes.
- Connector X2 (TCP/IP communication #2): interface for the same purposes than X1.
- Connector X4 (auxiliary TCP/IP communication for the CSM): interface with the CSM running on the ULTIMET ADVANCED.

Remark: IP addresses of TCP/IP communication #1 and #2 must not belong to the same sub-network. TCP/IP communication #1 and #2 are compatible with Ethernet speeds 10 Mbits/s, 100 Mbits/s and 1 Gbits/s while auxiliary TCP/IP communication dedicated to the CSM supports only 1 Gbits/s.

This chapter will focus on the installation and configuration of the first two TCP/IP communication interfaces. Refer to chapter D (page 149) for details concerning the installation and configuration of the auxiliary TCP/IP communication for the CSM.

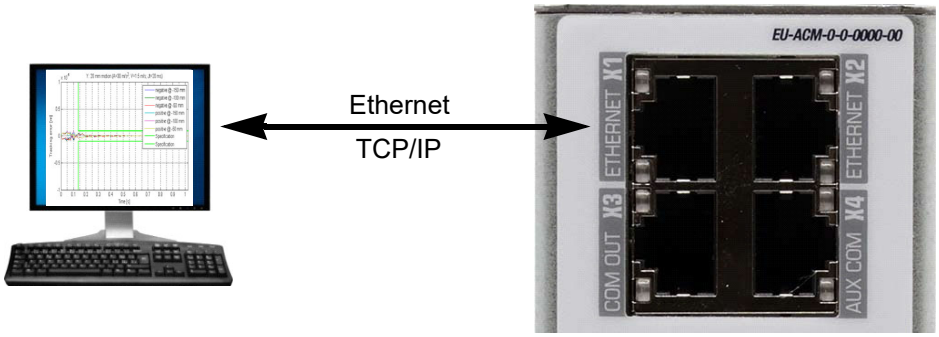
The ULTIMET ADVANCED also includes a cabled PCIe Gen2 (2.5 Gbps) interface via a PCIe x4 connector (X7) on the front panel. A cable and adapter board for a PCIe slot on the host PC are available with the “Starting kit” provided by ETEL (ordering information can be found in the ULTIMET ADVANCED “Hardware Manual”).

For more details related to the different interfaces available on the ULTIMET ADVANCED, please refer to the ULTIMET ADVANCED “Hardware Manual”.

3.1 TCP/IP connections

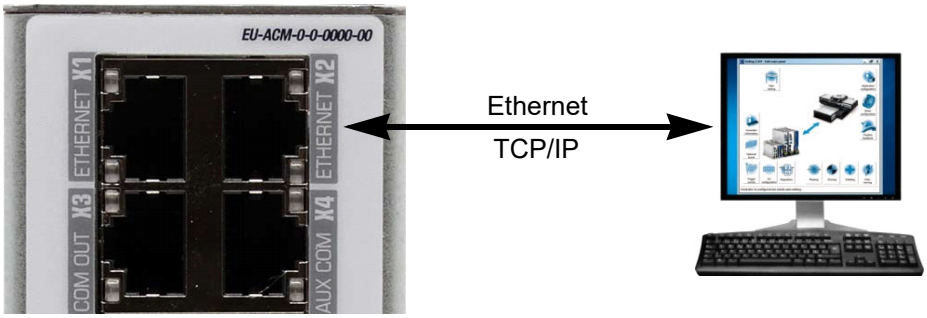
3.1.1 User application

This interface is intended for the communication between the user application and ULTIMET ADVANCED. The IP address can accept two connections, providing altogether two user application TCP/IP interfaces with the ULTIMET ADVANCED.



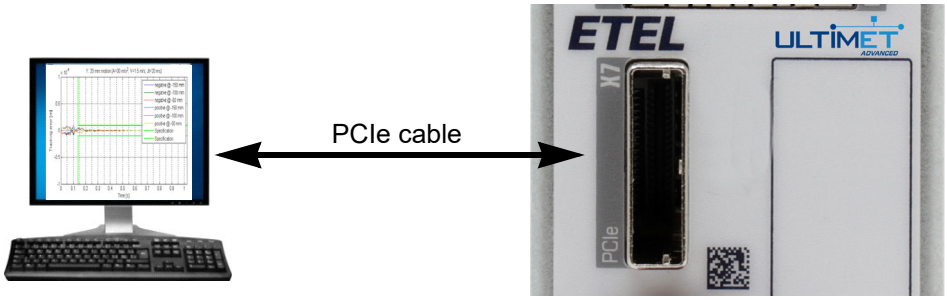
3.1.2 Commissioning

This interface is intended for commissioning purposes via ComET. The IP address can accept two connections, providing altogether two commissioning TCP/IP interfaces with the ULTIMET ADVANCED.



3.2 PCIe connection

This interface is intended for the communication between the user PC application and the ULTIMET ADVANCED. The communication latencies experienced with this interface are lower compared to a TCP/IP connection, therefore, its usage is recommended for applications with tighter real-time requirements.



The ULTIMET ADVANCED board is installed in a rack together with the AccurET controllers. As a consequence, the PCIe connection requires a PCIe x4 Gen2 host board for PC and an x4 PCIe cable. This PCIe host board does not require any third-party driver installation.

3.2.1 PCIe driver installation

To install the ULTIMET ADVANCED driver, please refer to “UltimET PCI-PCIe.pdf” file in EDI or ComET documentation directory.

3.2.2 ULTIMET ADVANCED boot up

3.2.2.1 General considerations

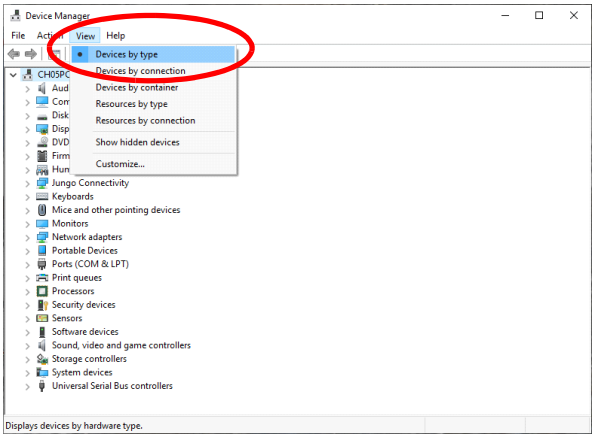
WARNING: The ULTIMET ADVANCED board must be connected to the PC host board and switched on before booting up the PC, otherwise the PC will not (and never) detect the ULTIMET ADVANCED board. This requirement derives from the PC’s PCIe devices enumeration procedure.

When the system is up and running, it is allowed to reboot the ULTIMET ADVANCED. However, this will make the PC lose track of the board, so after an ULTIMET ADVANCED reboot (power cycle or software), it is required to update the PC’s PCIe configuration.
Basically the device needs to be cleared from the PC’s list and then the PCIe devices rescanned.

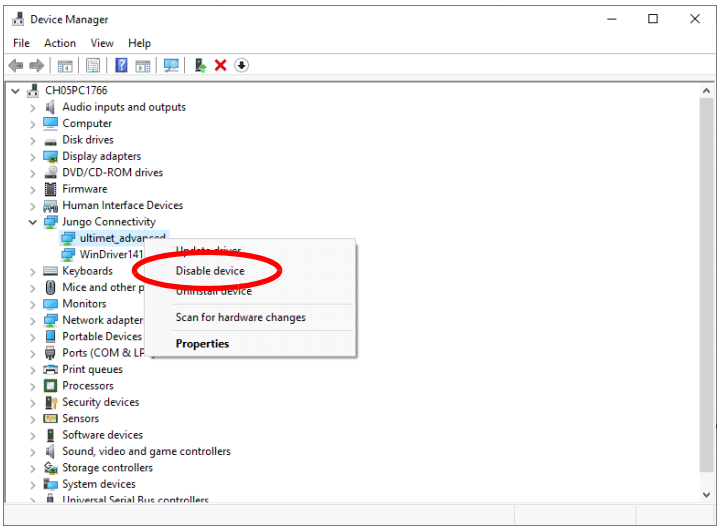
On Windows OS, these actions can be done
- Either with the “Device Manager” panel,
- Or via the Microsoft Device Console (Devcon.exe) tool. Devcon.exe and its sources are accessible on Microsoft web site (https://docs.microsoft.com/).

3.2.2.1.1 UA “reconnection” update using the Device Manager

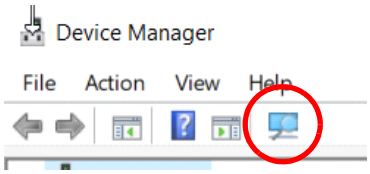
- Open the “Device Manager”.
- Check in the “View” menu that “Devices by type” is selected.



- On the devices tree, open the “Jungo Connectivity” element and then select and right click on the “ULTIMET ADVANCED” device.



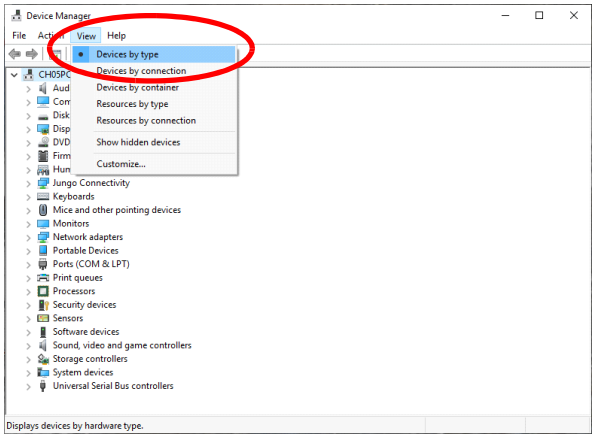
- Select “Disable Device”.
- Proceed again with the same previous steps but this time select “Enable Device”.
- Rescan the PCI devices by clicking on the “Action” menu and “Scan for hardware changes”:



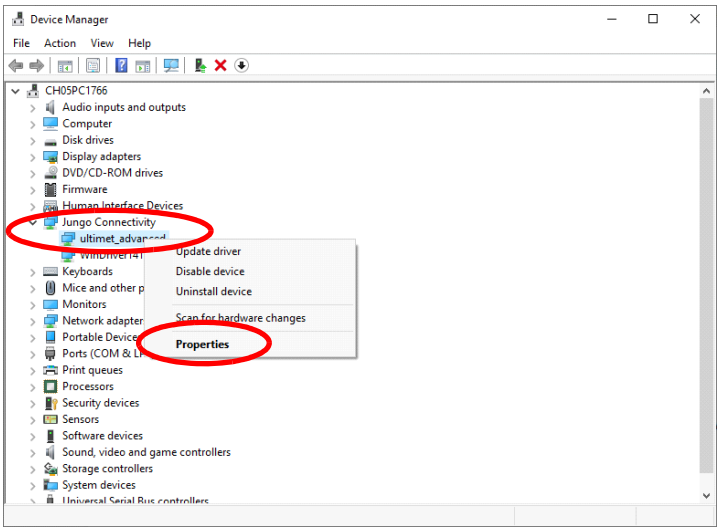
3.2.2.1.2 UA “reconnection” using the Microsoft Device Console

Using the Microsoft Device Console consists in writing a little script to be launched after each ULTIMET ADVANCED reboot (power cycle or soft reboot). This script needs to use the ULTIMET ADVANCED’s PC “parent device”. To find this parent device:

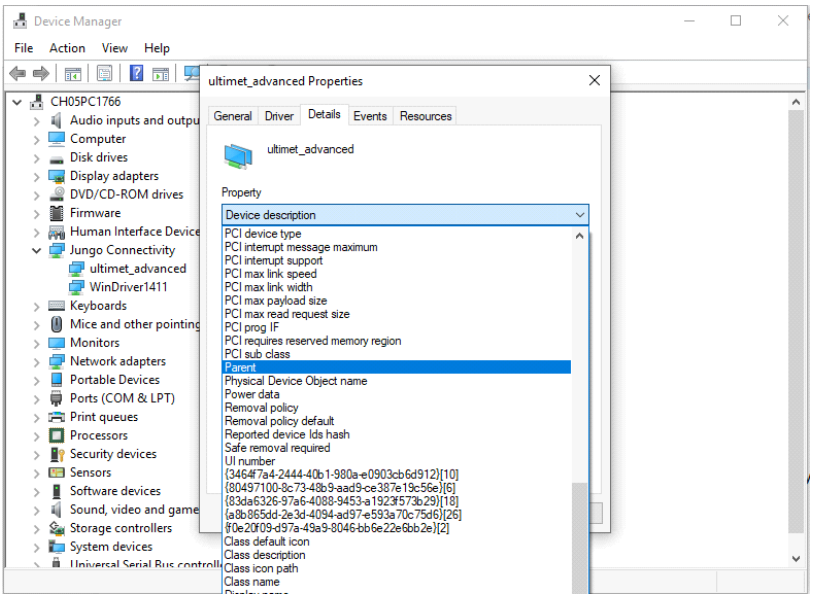
- Open the “Device Manager”.
- Check in the “View” menu that “Devices by type” is selected.



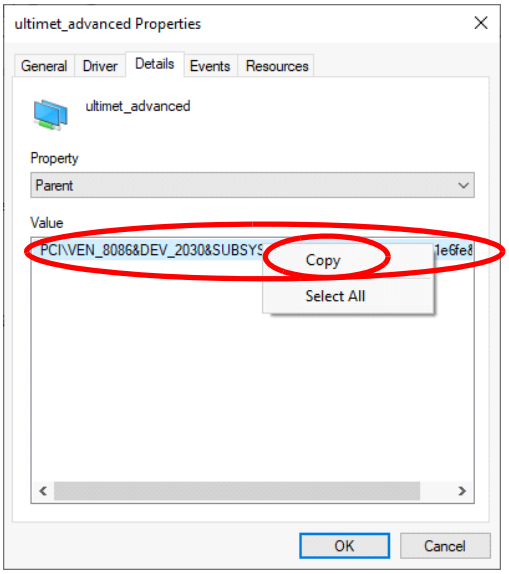
- On the devices’ tree, open the “Jungo Connectivity” element and then select and right click on the “ULTIMET ADVANCED” device and select “Properties”.



- Activate “Details” tab and select “Parent” property.



- Copy the value that describes parent hardware ID.



E.g: parentHwId: *PCI\VEN_8086&DEV_2030&SUBSYS_07381028&REV_04\3&2411e6fe&0&00*

Now, the following Windows command line script can be used to proceed to the configuration update, replacing `parentHwID` (keep the “”) with the value copied from above.

```
set currentDir=%CD%
cd C:\Program Files\MSDevCon
devcon restart @"parentHwID"
TIMEOUT 2
devcon rescan
cd %currentDir%
```

Run this command file after each reboot (power cycle or soft reboot) of the ULTIMET ADVANCED.

3.2.2.2 Special considerations when using the JTAG cable

A JTAG cable between the PC and the ULTIMET ADVANCED board is sometimes used, as described in §19.

To simplify the debugging over JTAG, ETEL provides in the starting kit a JTAG cable adapter with the reset pin unconnected. Thanks to this adapter, there is no considerations to take into account when using JTAG cable. However, if a standard JTAG cable is connected to the ULTIMET ADVANCED, it must be removed before the PC boot up. Otherwise the JTAG restarts the ULTIMET ADVANCED during the PC initialization and the PCIe enumeration procedure does not detect the ULTIMET ADVANCED board. Because this initial detection fails, even a device restart and rescan as described in §3.2.2.1 will not make the ULTIMET ADVANCED visible to the PC.

So, if a JTAG cable is being used without ETEL JTAG extension cable:
- Ensure it is removed before ULTIMET ADVANCED and then PC reboot.
- Once the PC is up, plug the JTAG cable back in.
- This may reboot the ULTIMET ADVANCED board. In which case, run the manual steps described in §3.2.2.1.1 or run the script written as per §3.2.2.1.2.

4 Configuration

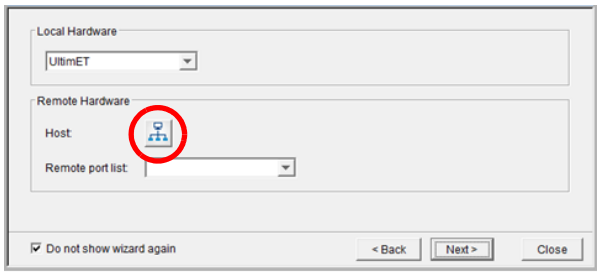
4.1 TCP/IP connection

The configuration of a TCP/IP connection is done via the ComET software. ComET is a user-friendly software interface developed by ETEL to set up and monitor the operation of the UltimET (and the associated position controllers). Users not familiar with this software should first read the 'ComET User's Manual'.

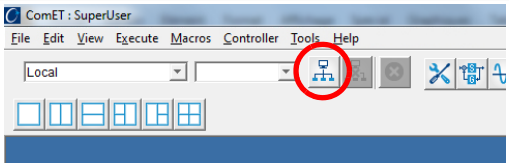
To configure the TCP/IP connection the IP address of the ULTIMET ADVANCED must be specified for the connector(s) used for the communication. Each connector supports two TCP/IP connections.

First of all, the user must establish the communication with the ULTIMET ADVANCED. To do so:

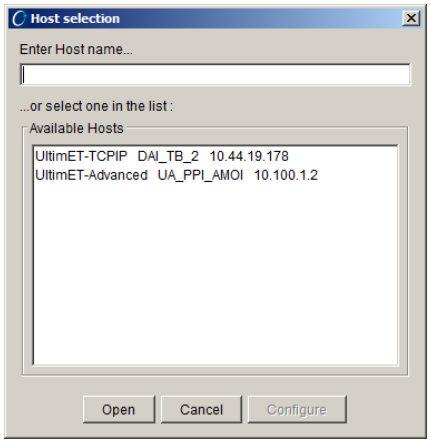
1. Double-click on the shortcut of your ComET.
2. Click on the 'Remote hardware' icon in order to find the ULTIMET ADVANCED connected to the network.



or



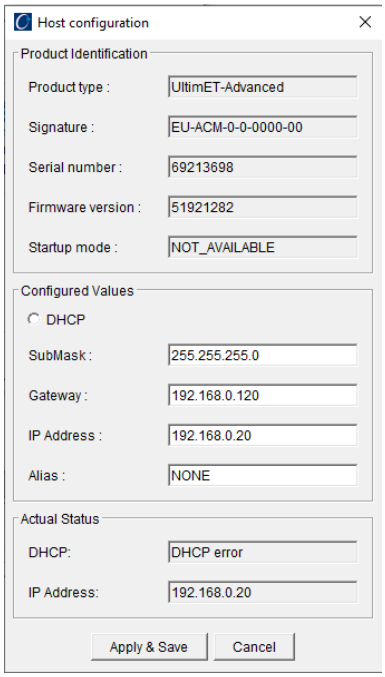
3. The Host selection window pops up displaying the ULTIMET ADVANCED connected to the network with the corresponding IP address. If one ULTIMET ADVANCED is interfaced through both connectors X1 and X2, there will be two entries listed, one for each IP address associated to each connector. In the example below, only one connector is being used.



The user can now select the desired ULTIMET ADVANCED TCP/IP connection to communicate with. The Host selection window will display both IP addresses configured for the two connectors X1 and X2 but the connection will be possible only with the one on the same sub-network as the PC running ComET.

Remark: By default the ULTIMET ADVANCED TCP/IP is configured with the IP address 192.168.0.20 for connector X1 and 192.168.1.20 for connector X2, both without DHCP enabled.

4. Once selected, the user can click on 'Configure' to change the network parameters of the TCP/IP connection. The following window is displayed:



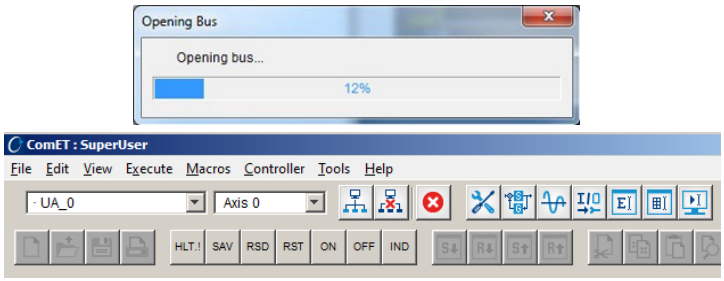
The 'Gateway' field allows the user to enter the gateway IP address to be connected to another sub-network. It must be on the same subnet as the IP address itself.

If a change has been done, the user must click on 'Apply & Save' to save the new parameters and then the ULTIMET ADVANCED must be rebooted to take the new parameters into account.

With the DHCP server option enabled, the ULTIMET ADVANCED 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 ADVANCED 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 1 Hz. After 60 seconds, if no DHCP server answers, the ULTIMET ADVANCED generates an error and uses the fixed IP address saved in flash or the default one. It is then possible to change the network configuration with ComET (which is not possible during the DHCP process). If a change has been done, the user must click on 'Apply & Save' to save the new parameters.

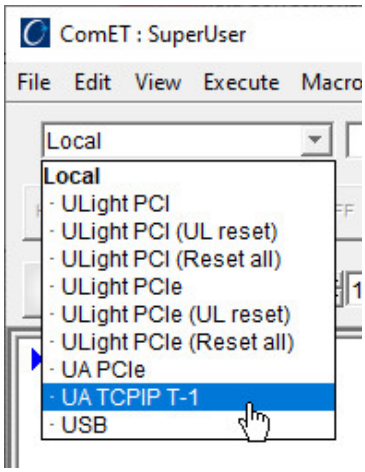
5. Now, click on 'Open' in the 'Host selection' window.

The communication with the ULTIMET ADVANCED via TCP/IP gets established:



Once the communication has been established, the user can have access to all the tools like e.g. the 'Scope', 'Terminal', 'Editor', etc. Refer to the 'ComET User's Manual' for more information.

Remark: Instead of executing the above-described procedure each time the user wants to use the ULTIMET ADVANCED, it is possible to create a shortcut in order to obtain "the" ULTIMET ADVANCED directly in the 'remote hardware' window (as depicted in the next picture).



Close ComET. Open the 'port properties' file present in the 'bin' folder of ComET. It is recommended to make a copy of the 'port properties' file as a wrong manipulation could prevent ComET from starting.

1.Duplicate the last paragraph of the 'port properties' file.

2.Increment (by 1) the value present immediately after **'port.'** in this new paragraph.

3.After the **'port.#.name = UA TCPIP T-1'** line, write (for example) **'ULTIMET ADVANCED'.**

4.After the **'port.#.driver ='** line, write **'ETN://192.168.0.20:1129'.** (The IP address can be different from the one mentioned above as it depends on the user's configuration).

5.The value present at the end of the **'port.#.accept.0.driver = ETN:#'** line must be replaced by a value above 1140 and different from the one present in the same line of the other paragraphs of the 'port properties' file.

The IP addresses configured for the ULTIMET ADVANCED connectors can be read back using the monitoring registers ***M205** and ***M206**.

M	Name	Comment
***M205**	Ethernet X1 IP address	IP address set for connector X1
***M206**	Ethernet X2 IP address	IP address set for connector X2

The MAC addresses that identify these two connection points can be read in the monitoring registers ***M203** and ***M204.** The MAC addresses are set at production time and cannot be changed.

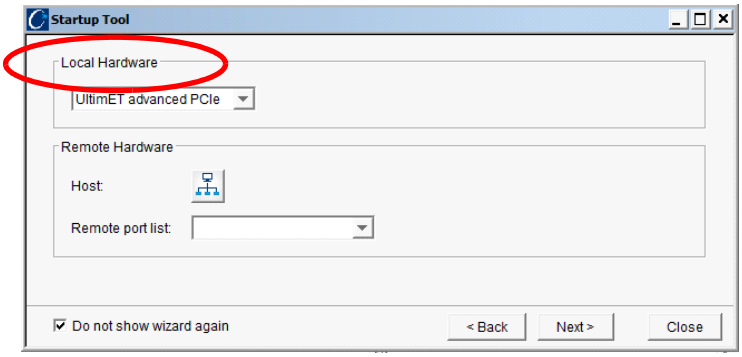
M	Name	Comment
***M203**	Ethernet X1 MAC address	MAC address set for connector X1
***M204**	Ethernet X2 MAC address	MAC address set for connector X2

4.2 PCIe connection

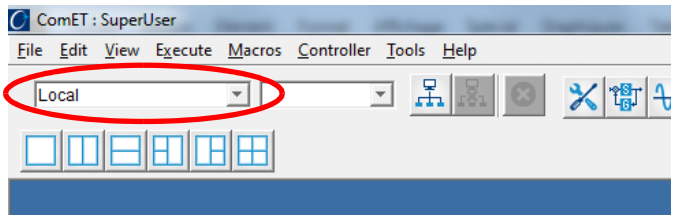
There is no need for any specific configuration prior to using the ULTIMET ADVANCED PCIe connection. When using ComET, the communication must be properly established, otherwise the tools cannot be used. To do so, select 'ULTIMET ADVANCED' in the 'Startup tool' or 'local' menu ('ULTIMET ADVANCED PCIe').

Remark: The ULTIMET ADVANCED must already be powered on when the host PC is started, otherwise the Windows OS will not detect the PCIe device and load the corresponding device driver. In such cases, any attempt to establish the communication with the ULTIMET ADVANCED via the PCIe interface will fail.

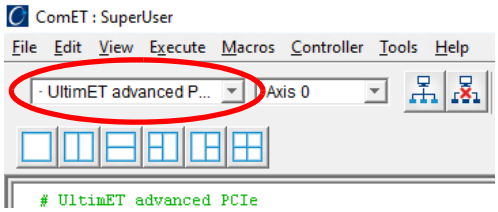
The ComET versions compatible with the ULTIMET ADVANCED PCIe connection start at 4.22A.



or



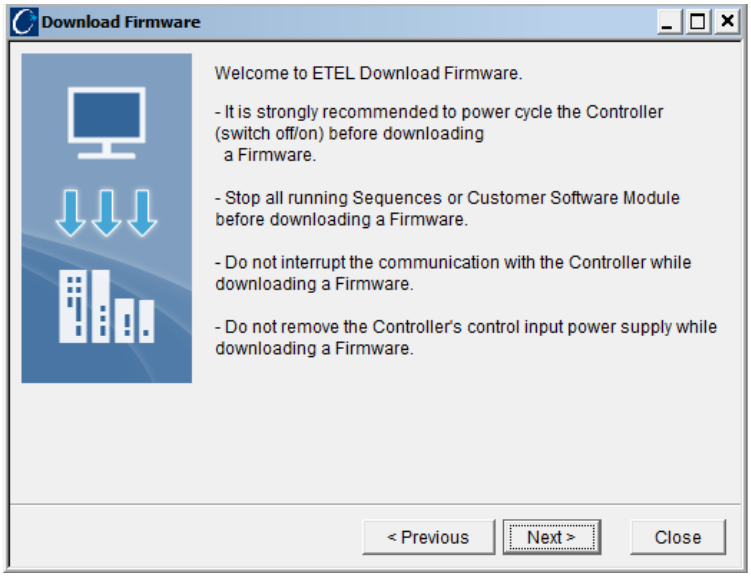
When the communication is established, 'ULTIMET ADVANCED PCIe' appears in the top left corner:



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

4.3 Firmware download

To download a new firmware to the ULTIMET ADVANCED, click on 'Download/Upload' in the menu bar and then on 'Download Firmware'. From now on, follow step-by-step the dialog boxes displayed by the 'Download Wizard' tool.



Remark: Firmware download is not available when connected through PCIe or through ETND.
The name of the firmware file must have the .far extension. Any ESM, if used, is included in the firmware far file. For further information, refer to §12.

If there is a problem during the download of the firmware (e.g. communication broken, main power line failure…), ULTIMET ADVANCED can be rebooted and will run its Recovery firmware (refer to §28.1). Then a new firmware download can be done again.

4.4 ULTIMET ADVANCED reset

It is sometimes required to reboot the ULTIMET ADVANCED, 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 ADVANCED 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 ADVANCED 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 UTIMET ADVANCED connected by PCIe, it will be necessary to have the host PC rescan the PCI devices (refer to §3.2.2 to find out how to do this) after the reboot, before the application can connect again.

THIS PAGE IS INTENTIONALLY LEFT BLANK

Chapter C: Functionalities

5 General commands & registers syntax

All the features of ULTIMET ADVANCED are handled via commands and/or registers. These can be accessed via the ComET terminal or using either generic or dedicated EDI functions. The following two paragraphs describe the general syntax for commands and register accesses from the ComET terminal.

Using the feature via dedicated EDI functions is described in the related paragraph. Refer to the EDI “User’s Manual” for the description of the generic EDI functions.

5.1 Commands

The «*» sign in front of a command identifies the ULTIMET ADVANCED 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: **[.<axis>]** or **[= <P1>]**) are **optional**. Not all commands use them.

*****
The command is sent to the ULTIMET ADVANCED.

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

.<axis>
Number of axis or group of axes 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 individual axes numbers in brackets, separated by commas.

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

Possible values:

- Immediate value in a format that depends on the command.
- Value contained in any local (ULTIMET ADVANCED) or remote (axis) register. Refer to §5.2.1.1 to know how to specify a register. The register specification must be consistent with the type expected by the command: if parameter P2 is expected to be a 64-bit float, e.g., then the register specification must refer to a 64-bit float, i.e. the register type name ends in ‘D’ (XD, LD,...).

Remark: If the command and its (or some) parameters refer to different axes, the ULTIMET ADVANCED 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 1 parameter (P1=1), sent to UltimET and axis 1.
*SAV.(1,2,8) = 1;  // Command with 1 parameter (P1=1), sent to UltimET and axes
                   // 1, 2 and 8.
*JMP.1 = 10, 1;    // Command with 2 parameters (P1=10 and P2=1), sent to
                   // UltimET and axis 1.
```

Remark: If at least one parameter (P1,...) of a command is wrong (bad format for example), an error (code 1159) occurs.

5.2 Registers

5.2.1 Registers group

The **registers** are accessible to the user, they store all the ULTIMET ADVANCED'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 ADVANCED and some others are common to both ULTIMET ADVANCED and AccurET controllers. There are 8 main types of registers, 3 are basic (often or always used) and 5 are advanced (for specific applications only).

5.2.1.1 Basic registers

K, for **parameters** — They define the configuration of UltimET and the setting of its features. Refer to §30. for the complete lists of K parameters. The parameters can be either common with the AccurET controller or dedicated to the ULTIMET ADVANCED.

M, for **monitorings** — They are exclusively used to monitor the ULTIMET ADVANCED'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 §31. for the complete lists of M monitoring registers.

X, for **user variables** — They are variables that the user may use freely for programming. Each user variable's purpose is at the user's discretion. User variables can be both read and write at any time.

An extra letter in the register name indicates the type of the associated value: 'L' for 'Long' (64 bits integer), 'F' for 'Float' (32 bits float) and 'D' for 'Double' (64 bits float). This rule can be applied to all registers. The register names become 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. The K and M registers in range 0 to 511 have the same meaning as for the AccurET controllers.

	Register	Integer	Integer 64 bits	Float	Double
Common to AccurET position controller	K	0 ≤ K ≤ 511	0 ≤ KL ≤ 511	0 ≤ KF ≤ 511	0 ≤ KD ≤ 511
	M	0 ≤ M ≤ 511	0 ≤ ML ≤ 511	0 ≤ MF ≤ 511	0 ≤ MD ≤ 511

The K and M registers beyond 511 and the other registers are specific to the ULTIMET ADVANCED.

	Register	Integer	Integer 64 bits	Float	Double
Specific to ULTIMET ADVANCED	K	512 ≤ K ≤ 1024	512 ≤ KL ≤ 1024	512 ≤ KF ≤ 1024	512 ≤ KD ≤ 1024
	M	512 ≤ M ≤ 1024	512 ≤ ML ≤ 1024	512 ≤ MF ≤ 1024	512 ≤ MD ≤ 1024
	X	0 ≤ X ≤ 511	0 ≤ XL ≤ 255	0 ≤ XF ≤ 511	0 ≤ XD ≤ 255

In the 'Terminal' of ComET the value formats are displayed and must be typed in as follow:

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

Remark: An **alias** is a more user-friendly name to refer to a K parameter K or an M monitoring register like DIN for Digital INput. Their syntax is identical to the one of the corresponding register. Refer to §30. and §31. to know the list of the alias.

Example: *DOUT=0xF0 is the same as *K171=0xF0

5.2.1.2 *Advanced registers*

L, for **LKT** — Also called ***look-up table registers.*** They are general purpose registers for temporary logs or measurements. They are available for read and write.

T, for **trace** — They allow the acquisition of registers X, K, M and L of the ULTIMET ADVANCED. They are used by the 'Scope' of ComET (0≤T≤16383). They are available for read and write.

Y — Also called ***advanced user registers.*** They are used to change a target at any time in a PVT (Position-Velocity-Time) movement, but are free to be used at will when not performing interpolated movements. They are available for read and write.

W and **R** — These are the equivalent of the K and M registers (respectively) but specifically dedicated to a given ETEL Software Module (ESM). They are only available when an ESM has been loaded and they depend on the ESM itself. Refer to §12 for more information. W registers are available for read and write. R registers are read only.

	Register	Integer	Integer 64 bits	Float	Double
Specific to ULTIMET ADVANCED	L	0 ≤ L ≤ 8191	Not available	Not available	0 ≤ LD ≤ 8191
	T	0 ≤ T ≤ 16383	0 ≤ TL ≤ 16383	0 ≤ TF ≤ 16383	0 ≤ TD ≤ 16383
	Y	0 ≤ Y ≤ 256	Not available	Not available	Not available
	W	0 ≤ W ≤ 511	0 ≤ WL ≤ 511	0 ≤ WF ≤ 511	0 ≤ WD ≤ 511
	R	0 ≤ R ≤ 511	0 ≤ RL ≤ 511	0 ≤ RF ≤ 511	0 ≤ RD ≤ 511

5.2.2 Register specification (ComET terminal)

Reading the value of a register from the ComET terminal is done by just typing its specification. To assign a value to a register, append an assignment operator – the simplest being "=" – to its specification. Refer to §5.2.3 for register assignment operations and to the EDI "User's Manual" for more information about accessing registers from an application. The syntax describing a register, is as follows:

Syntax:

[*]<register type><register index>[:<depth>][.<axis>]

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

The «*» sign in front of a register identifies the ULTIMET ADVANCED 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 assign a value to the same register on several addressees. It is not allowed for reading a register value.

<register type > — Defines the type of register used:

Possible types:

- **K, KF, KL, KD, M, MF, ML, MD, X, XF, XL, XD, T, TF, TL, TD, W, WF, WL, WD, R, RF, RL, RD, L, LD and Y.**

<register index > — Defines which register of **type** is referenced. It is a number (within the range of the given type).

[:<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 depends on the register type:

- 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 **3** if the register type is **L**

- Integer from **0** to **9** if the register type is **LD**
- Integer from **0** to **3** if the register type is **Y**

For the other register types the number of depths depends on the register number. It is specified in the paragraph describing the feature using the register.

[.<axis>]

Example:

```
*K121:1;        // Read the value of the parameter K121 of the ULTIMET ADVANCED at
                // the depth 1.
K121.3;         // Read the value of the parameter K121 of axis 3 at the depth 0.
```

5.2.3 Register assignment (ComET terminal)

Writing a value to a register is done by appending an assignment operator to a register specification described above. The syntax to change the value in a register is as follows:

Syntax:

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

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

[<operator>] Mathematical sign for arithmetic and logical operations. It is available only for **K** and **X** registers.

Possible values:

- + 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 AND NOT (not available for Float 32 and 64 bits).
- |~ bitwise OR NOT (not available for Float 32 and 64 bits).
- ^~ bitwise XOR NOT (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).

= mandatory assignment sign.

<P1> Value of the register.

Possible values:

- Immediate value in the format corresponding to the register data type (Integer, Long, Float or Double).

Remark: For register types K, and W a **maximum value** and a **minimum value** are defined. If a value higher than the maximum value or a value smaller 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. The default value is the same for all depths.

- Value contained in another register. It can be taken from any axis. Specify as described in §5.2.2.

- All values in the different depths are expressed in the same unit. The unit depends on what the register represents.

Remark: Like for commands, when assigning a register with another on the same axis, the ULTIMET ADVANCED will send the command as is; otherwise it will first fetch the actual value and send it in the assignment command.

Example:

```
*K177 = 3;            // The value 3 is assigned to depth 0 of parameter K177 of
                      // the ULTIMET ADVANCED.
*K177 = X21.2;        // The value of the user variable X21 of the axis 2 is
                      // assigned to depth 0 of parameter K177 of the ULTIMET
                      // ADVANCED.
*K177.(1,4,5) = 3;    // The value 3 is assigned to depth 0 of parameter K177 of
                      // the axes 1, 4 and 5 and to the ULTIMET ADVANCED.
*K177.! = 3;          // The value 3 is assigned to depth 0 of parameter K177 of
                      // all axes and to the ULTIMET ADVANCED.
*X0.0 = 1050;
*X0.0 %= 100;         // Modulo 100 => X0 = 50
*XL10 = 10L;
*LD25 = 2.5D;
```

6 Registers handling

6.1 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	***CLX[.<axis>] = <P1>**		
Description	Clears all user variables (X, XF, XL, XD, Y register depending on <P1>)		
Parameter	**Description**	**Value**	**Description**
<P1>	Clears registers X and Y	1 (bit#0) 2 (bit#1) 4 (bit#2) 8 (bit#3) 16 (bit#4)	Clears X user variables. Clears XF user variables. Clears XL user variables. Clears XD user variables. 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 = 2;      // The *XF4 variable (as well as all the others XF variables)
               // contains the value 0.
```

6.2 SET_RANGE command

The **SET_RANGE** command sets in one block n values into X, XL, XF, XD or L registers. For X, XF and L registers, the maximum number of values is n = 126. For XL and XD registers, the maximum number of values is n = 99. It sets in the specified type of register starting from a specified index at a fixed depth (<P1> value) the values given as parameters of the command.

Command	***SET_RANGE = <P1>, ..., <Pn+1>**
Description	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.
Parameter	**Description**
<P1>	Type of register, first index and depth from where the different values have to be written.
<P2>	Value 1 to be written in the register specified in parameter 1.
...	...
<Pn+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.
*SET_RANGE = *X12:3, 100, 101, 102, 104;   // Sets *X12:3=100, X13:3=101,
                                           // *X14:3=102 and *X15:3=104.
```

6.3 Set / reset bit(s) management

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

Command	***CH_BIT_REG32.<axis> = <P1>, <P2>, <P3>**
Description	Allows writing any bits (set bits and/or clear bits) given by mask <P2> with defined value <P3> of any 32-bit integer register <P1> without modifying the other bits in the register.
Parameter	**Description**
<P1>	Destination register

<P2>	Mask of the bits that can be modified
<P3>	Mask value of the bits

This will do <P1> = (<P1> & ~<P2>) | (<P3>&<P2>)

Example:

A CSM thread (e.g. 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`

Another CSM thread (e.g. 2) controls bits 2,and 3 of K177:0.<axis>.

- 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`

The CH_BIT_REG32 command is thread safe, i.e there will be no conflicts between threads.

6.4 Set several registers

The **SSR** command (**S**et **S**everal **R**egisters) allows the user to assign up to 6 registers in the same command.

Command	***SSR[.<axis>] = <P1>, <P2>, [<P3>,...<P12>]**
Description	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).

Example:

```
*SSR = 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.

6.5 CHKDISTGRT command

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

Command	***CHKDISTGRT = <P1>, <P2>, <P3>**
Description	Checks the values in the registers specified in the first 2 parameters are separated by the value given in the third.
Parameter	**Description**
<P1>	Defines the first register to compare
<P2>	Defines the second register to compare
<P3>	Defines the minimum distance between the two registers

The command compares the values of the first two parameters which must represent register specifications and generates the error 5800 if the absolute value of the difference 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. It starts immediately after the command is issued. Specifying an immediate null value as the third parameter disables the check.

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

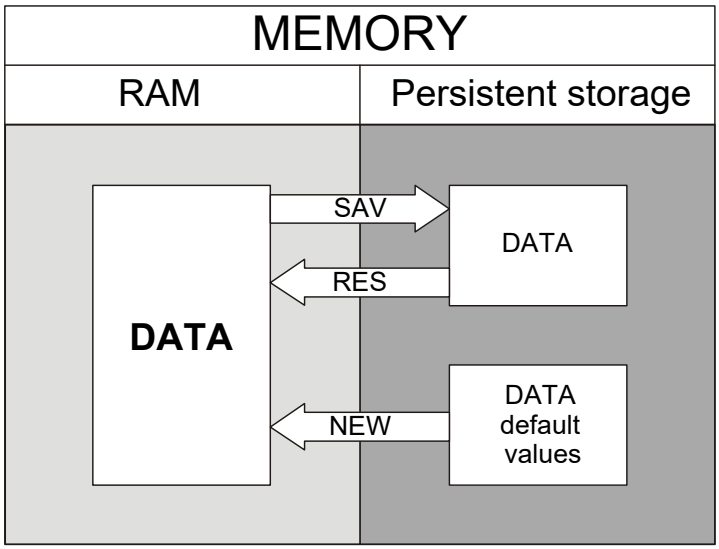
Example:

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

6.6 Saving and restoring the registers

The ULTIMET ADVANCED has **volatile runtime RAM memory** and **persistent storage memory**. The RAM is a volatile memory which is erased each time the ULTIMET ADVANCED is switched off, whereas the permanent storage is a non-volatile memory and the data stored inside are not lost when the ULTIMET ADVANCED is switched off. **All calculations and operations done by the controller are realized with the values present in the RAM**.

SAV, RES and NEW commands allow the user to transfer data from the RAM to the persistent storage and vice-versa.



The **SAV** command (**SAV**e) **saves into the persistent** memory the ULTIMET ADVANCED's data (K, W, X, L and Y registers). The type of saved data is defined by the parameter's value of the command.

Command	***SAV[.<axis>] = <P1>**		
Description	Saves the current registers (K, W, X, Y and L) in persistent storage in function of <P1>		
Parameter	**Description**	**Value**	**Description**
<P1>	Register selection	0	Saves L look-up tables, K and W parameters, X user variables and Y registers in persistence storage
		1	Saves L look-up tables in persistent storage
		2	Saves K parameters, W parameters, X user variables and Y registers in persistent storage
		3	Saves K parameters in persistent storage
		5	Saves X user variables and Y registers in persistent storage
		6	Saves L look-up tables in persistent storage
		10	Saves ESM W parameter in persistent storage

The **RES** command (**RES**tore) restores into the **RAM** memory the ULTIMET ADVANCED's data (K, W, X, L and Y registers) stored in persistent storage. The type of restored data is defined by the parameter's value of the command.

Command	*RES[.<axis>] = <P1>		
Description	Restores the persistent storage saved registers (K, W, X, Y and L) in ram memory according to <P1>		
Parameter	**Description**	**Value**	**Description**
<P1>	Register selection	0	Restores L look-up tables, K and W parameters and X user variables and Y registers from persistent storage to ram memory
		1	Restores L look-up tables from persistent storage to ram memory
		2	Restores K parameters, W parameters, X user variables and Y registers from persistent storage to ram memory
		3	Restores K parameters from persistent storage to ram memory
		5	Restores X user variables and Y registers from persistent storage to ram memory
		6	Restores L look-up tables from persistent storage to ram memory
		10	Restores ESM W parameter from persistent storage to RAM

The **NEW** command **reloads** in the **RAM** the default values of K and W registers and **clears** the X, L **and Y** registers.

Command	*NEW[.<axis>] = <P1>		
Description	Restores the default registers values (K and W), clears registers (X, Y and L) in ram memory according to <P1>.		
Parameter	**Description**	**Value**	**Description**
<P1>	Register selection	0	Clears X user variables, Y registers, sets default values for K and W parameters in ram memory
		2	Sets default values for K and W 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
		10	Clears ESM W registers in ram memory

After executing RES and NEW commands, current memory values are replaced by the values read in the micro-SD card and are definitively lost. Similarly, the SAV command overwrites the values contained in the micro-SD card with those contained in memory.

Remark: When the ULTIMET ADVANCED is **switched on**, data is automatically restored, like with a ***RES = 0** command (so that it is not necessary to do it manually), apart from ESM registers which might be handled slightly differently. Refer to §12 for the more information.
All register depths are saved and read when SAV, RES and NEW commands are executed.

It is possible to assign a version number for the registers of a given type used in each controller. The configuration of the version number is done in the ComET editor before the download of 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 register ***M282** allows the user to get the register types versions.

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

7 WAIT Commands

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.

A wait command sent to an axis will block any other normal command sent to a group of axis that includes that axis, regardless of the application interface or CSM thread that sends these commands. Currently, this is a known issue, a wait command sent to the ULTIMET ADVANCED will block all the application interfaces (PCIe, TCP/IP and CSM).

There is a command that allows to clear a pending wait (refer to §). This command has to be sent as an urgent command (otherwise it will itself be blocked by the wait it is trying to clear).

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

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

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

Warning: <P1> and <P2> must belong to the node executing the 'wait' command. In addition, this command will read only once the parameters on a node at the beginning of the waiting. Therefore, it will not see any changes of the register's value.

For example, `*WPG = X1.1,12` is forbidden: the command 'wait' is done on the ULTIMET ADVANCED but the variable X1.1 is on node 1.

7.1 WTT command

The **WTT** command (**W**ai**T** **T**ime) makes a pause in the execution. The pause duration is set with the parameter of the command.

Command	***WTT[.<axis>] = <P1>**
Description	Wait the time defined by <P1>, in increments of 100 µs.
Parameter	**Description**
<P1>	Time

Remark: Refer to §7.3 to know how to clear a 'wait' command.

Example:

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

7.2 WTM commands

The **WTM** command (**W**ai**T** for end of **M**ovement) allows the user to wait for the end of the interpolated movement before going on with the execution.

Command	***WTM[.<axis>] = <P1>**
Description	Waits for the end of interpolated movement of the group defined by <P1>. This condition is fulfilled when the current movement is finished and the interpolation command buffer is empty.
Parameter	**Description**
<P1>	Interpolation group

Remark: Refer to §7.3 o know how to clear a 'wait' command.
If the motor is not moving when the WTM command is sent, it is ignored.

Example:

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

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

WBS (**W**ait **B**it **S**et) and **WBC** (**W**ait **B**it **C**lear) commands test one or several bits of X, K or M registers and return if and when 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	***WBS[.<axis>] = <P1>, <P2>**
Description	Waits for the specified bits (which have their mask value included in <P2>) of the specified register <P1> to be all set.
Parameter	**Description**
<P1>	Value or register to test
<P2>	Bit mask

Command	***WBC[.<axis>] = <P1>, <P2>**
Description	Waits for the specified bits (which have their mask value included in <P2>) of the specified register (<P1>) to be all cleared.
Parameter	**Description**
<P1>	Value or register to test
<P2>	Bit mask

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 and of the ULTIMET ADVANCED to be at 1.
*WBC = *X3:2, 0x0F;     // Wait for the bits 0, 1, 2 and 3 of the ULTIMET ADVANCED
                        // variable *X3 of depth 2 to be at 0.
```

WSBS (**W**ait **S**tatus **B**it **S**et) and **WSBC** (**W**ait **S**tatus **B**it **C**lear) commands test one or several bits of M63 monitoring register and returns if and when 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	***WSBS = <P1>, <P2>**
Description	Wait for status (M63) bits set in several axes in the same time.
Parameter	**Description**
<P1>	Bits mask of M63 that must be set
<P2>	Mask of the nodes concerned by the wait status bit set

Command	***WSBC = <P1>, <P2>**
Description	Wait for status (M63) bits cleared in several axes in the same time.
Parameter	**Description**
<P1>	Bits mask of M63 that must be cleared
<P2>	Mask of the nodes concerned by the wait status bit clear

Remark: Refer to §7.3 to know how to clear a 'wait' command.

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; // 0x00200000: status bit "in window",
                           // 0x48L: node bits 3 and 6.
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;  // 0x01000000: bit "sequence running",
                           // 0xC: nodes bits 2 and 3.
DOUT.4 = 1;                // Activation of DOUT1 of the node 4.
```

7.4 Wait on values: WPL and WPG commands

The **WPL** (**W**ait **P**arameter **L**ower than) and **WPG** (**W**ait **P**arameter **G**reater than) commands 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 execution.

Command	***WPL[.<axis>] = <P1>, <P2>**
Description	Waits until <P1> becomes lower than <P2>.
Parameter	**Description**
<P1>	Value or register to test
<P2>	Reference value or register

Command	***WPG[.<axis>] = <P1>, <P2>**
Description	Waits until <P1> becomes greater than <P2>.
Parameter	**Description**
<P1>	Value or register to test
<P2>	Reference value or register

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 command returns immediately.
The value in <P2> is only taken into account when the WPL and WPG commands are executed.
Refer to §7.3 to know how to clear a 'wait' command.

7.5 Clearing a wait command

All the 'wait' commands described above temporarily stop the execution of other normal commands on the same application interface (that sent the 'wait' command).

The **CLRWAIT** (**CL**ea**R WAIT**) command allows the user to clear a pending 'wait' which is blocking the application interface and thus enables the execution to carry on. This command must be sent as an urgent command.

Command	**!*CLRWAIT=<P1>**
Description	Clear busy state due to a wait command. Currently – pending the resolution of a known issue - <P1> is ignored and the commands clears the wait command sent by the same application interface that sends this CLRWAIT command.

8 Status

8.1 ULTIMET ADVANCED controller status

The 'controller status', given by monitoring register **M63**, allows the user to see if the ULTIMET ADVANCED is in error, is executing a movement, is running a CSM thread, etc.

M	Comment
***M63**	Gives the ULTIMET ADVANCED status

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 §31. for more information)
16 to 18	-	-	Reserved
19	524288	Present	1=node present; 0=node not present
20	1048576	Moving	The ULTIMET ADVANCED 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 ADVANCED is executing a CSM thread. *M97 indicates which threads are running (refer to §20.1.1.2)
25	33554432	Ipol0 moving	The ULTIMET ADVANCED is executing an interpolated trajectory, group 0
26	67108864	Error	1=The ULTIMET ADVANCED 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 ADVANCED is executing an interpolated trajectory, group 1
30	1073741824	-	The TransnET connection has been interrupted
31	2147483648	-	The controller is executing one of the threads specified by *K297

The content of this monitoring register is updated each time a change occurs.

Remark: Refer to §31. for more information about the status given by the monitoring register M63.

The users status (bits 0 to 15) are a copy of the user status register *K177 parameter. Refer to §8.2 For the usage details of *K177.
The user can wait for a given state of some status bits using the general commands *WSBS and *WSBC (refer to §7.3).

8.2 User status parameter

Parameter ***K177** is available for the user to set bits that are reflected in the ULTIMET ADVANCED status word.

K	Comment
***K177**	Sets the user status

This provides a means for threads or different application interface applications to communicate, *K177 bits being set somewhere and tested elsewhere e.g. with *WSBS and *WSBC commands (§7.3).

8.3 Temperature

The temperature of the ULTIMET ADVANCED board processor can be monitored by reading the monitoring register ***M90**.

M	Name	Comment
***M90**	Device temperature	Depth 0: current temperature Depth 1: minimum recorded temperature Depth 2: maximum recorded temperature

8.4 Errors and warnings management

8.4.1 Error notification

When an error occurs on the ULTIMET ADVANCED, bit#26 of *M63 is set to one and the reason for the error is described by the code set in the monitoring register ***M64**.

M	Name	Comment
***M64**	Error code	Refer to §33. for the error codes list and their meaning.

The ULTIMET ADVANCED also keeps track of the TransnET nodes that are in error in the monitoring register ***ML630**.

M	Name	Comment
***ML630**	Bit mask of TransnET nodes in error	The nodes that are in error have the bit with their node number set to one in this register.

8.4.2 Reaction to errors

When an error occurs, the action of the ULTIMET ADVANCED can be programmed. Parameter ***K631** allows the user to choose between different behavior.

K	Value	Comment
***K631**	0 1 2	- Stops the interpolated movements, sends the emergency command !ERR to the interpolated nodes. - Stops the interpolated movements, sends the emergency command !HLO to the interpolated nodes. - Brakes the interpolated movements with a programmed deceleration, sends the emergency command !HLB to the interpolated nodes.

The purpose of emergency commands is to be able to stop and cut the power without waiting for the nodes whose movement is managed by the ULTIMET ADVANCED, i.e those involved in an interpolated movement.

The available commands are:

- !*ERR: sends the ERR command to all the interpolated nodes. This command triggers an error in the controller which opens the power bridge and displays “external error”.
- !*HLO: sends the HLO command to all the interpolated nodes; the controller opens the power bridge but does not display any error.
- !*HLB: sends the HLB command to all the interpolated nodes and the ULTIMET ADVANCED which slows the movement down to a stop. The deceleration of the axes belonging to the interpolation group is given by the ULTIMET ADVANCED parameters corresponding to the movement types. The controller does not display any error.

Remark: There is currently no direct programmable reaction to errors for a CSM. Refer to chapter D (page 149) for an example for how to implement this.

8.4.3 Errors propagation on TransnET

On all AccurETs and UltimET, there is a mechanism of errors propagation via TransnET. It is possible to define 4 independent error groups of axes.

This mechanism is managed by two parameters:

- Parameter ***K141** allows the user to define the group in which the error must be published when an error occurs on the ULTIMET ADVANCED:

K	Name	Comment
***K141**	Published error group mask	Gives the published error 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 register ***M141** allows the user to check the value of the error propagation register on the TransnET.

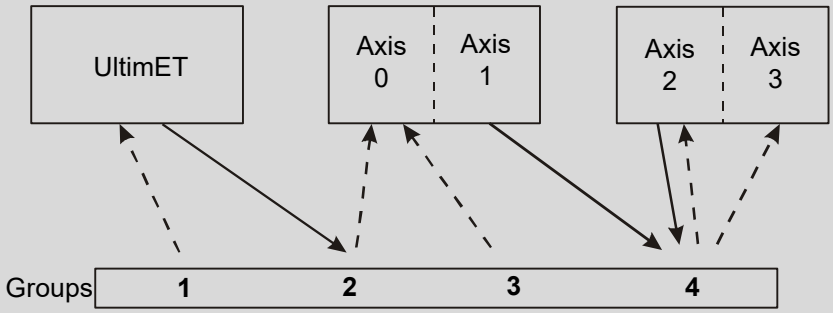
M	Name	Comment
***M141**	TransnET error groups status	Gives the value of the error propagation register.

- Parameter ***K140** allows the user to define which error group the ULTIMET ADVANCED 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**	Checked error group mask	Gives the checked error propagation group mask: bit#0 for group 1, bit#1 for group 2, bit#2 for group 3 and bit#3 for group 4.

It is not possible to power on an axis as long as one of the devices belonging to the same error group is in error. 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.

Example:



The setting corresponding to the above drawing is as follows:

```
// Errors put into groups 2 and 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
```

8.4.4 TransnET communication errors

The monitoring register ***M630** indicates if the communication with the nodes on TransnET is established. It is set to 0 at start up and the time needed for the bus to become ready and for ULTIMET ADVANCED to gather some information about the nodes connected, after which, during normal operation it is set to 1.

If the communication is broken (cable pulled out, too many transmission errors,…) it drops back to 0 until the issue is solved and the communication is setup once again.

M	Name	Comment
***M630**	TransnET operational status	=1 when the communication over TransnET is established. =0 when the communication is broken or in the process of being set up.

Another indicator of the TransnET health is the reference counter reflected in monitoring register ***ML228**. It is a value that is incremented by the ULTIMET ADVANCED each cycle in units of 25 µs when in normal operation.

M	Name	Comment
***M228** ***ML228**	Reference time counter	Reference time counter in unit of 25 µs. This counter is synchronized by the ULTIMET ADVANCED when TransnET is connected.

8.4.4.1 Isolated communication error

A 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 ADVANCED.
- Status: values of the last valid status are stored in the ULTIMET ADVANCED and transmitted to the application (PC or CSM) as long as the error persists.
- This error is indicated neither in the error bit of the status nor in *M64.
- The monitoring register ***M629** allows the user to know the number of errors that occurred on the TransnET.

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

8.4.4.2 Fatal communication error

If the frequency of the isolated errors is too high, the ULTIMET ADVANCED enters an error mode. The effect on the system is:

- The error bit of the ULTIMET ADVANCED's status is activated.
- *M64 stores 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 for the ULTIMET ADVANCED.

8.4.5 The *ERR command

The **ERR** (**ERR**or) command puts the ULTIMET ADVANCED in error. There is an optional parameter to the command: ***ERR[.<AXIS>] [=<P1>]**. It allows the user to set application errors through the CSM for example. In that case, <P1> **must be a negative value.**

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

Command	***ERR[.<AXIS>] [= <P1>]**
Description	Puts ULTIMET ADVANCED in error (error number (M64) = <P1>).

8.4.6 Resetting errors

The **RST** command enables the user to reset the ULTIMET ADVANCED and nodes' errors. If the ULTIMET ADVANCED 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 **'Communication error. TransnET error'** error (M64=1000) cannot be reset as long as the TransnET does not work properly.

Command	***RST[.<AXIS>]**
Description	Resets the error flag of ULTIMET ADVANCED

Example:

```
*RST;       // Reset the errors of the ULTIMET ADVANCED.
*RST.!;     // Reset all the nodes and ULTIMET ADVANCED's errors.
```

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

8.4.7 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 monitoring registers ***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 individually. The default value for this parameter is 0x0 and means no warnings are masked. When a bit of *K166 is set, the corresponding warning is masked and will appear neither in *M66 nor in bit#23 of *M63.

K	Name	Comment
***K166**	Warnings mask	Gives the bits masking the corresponding warning messages

Example:

```
*K166 = 0x80000;     // Mask the "breakpoint" warning
```

The monitoring register ***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**	Warnings flag	Bit field where each bit set indicates an active warning

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

M	Alias	Name	Comment
***M66**	WARCD	Warning code	Refer to §32. for the warning codes and their signification

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 a yellow circle on top of the 'UltimET' icon in its status bar. If there is also an error signaled, it is the red circle for the error that has the priority. The bit definitions of *M166 and *K166 are identical.

There are two types of warnings on ULTIMET ADVANCED:
- Contextual warnings: these warnings are automatically reset when the warning condition disappears.
- Persistent warnings: these warnings remain active once the warning condition has appeared, regardless of whether the condition disappears at a later stage. These warnings must be reset by the user with a **WRST** (**W**arning **R**e**S**e**T**) command. The *RST reset error command will also clear any existing persistent warning.

Command	***WRST**
Description	Reset remanent Warnings

The type of each warning is also shown in §32.

8.5 User LEDs

The LEDs on connector X6 (refer to "Hardware manual") are available for the user's purpose. These can be controlled with parameter ***K595**.

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

8.6 Firmware logs

The ULTIMET ADVANCED logs its errors, warnings and activities in log files located on the internal SD card. When the board starts, the firmware creates a new log file. When that log files reaches 10 MB, it is closed and a new one is created.
When the count of log files reaches 100, the oldest is deleted when the new one is created. These log files can be uploaded with ComET.

The information logged in these files is time stamped. But the ULTIMET ADVANCED board does not have an RTC (on-board, real-time clock) which would require a battery, then:

- If the ULTIMET ADVANCED is connected to a PC, either via TCP/IP or PCIe, connecting with EDI will set the ULTIMET ADVANCED's time with the PC time. The ULTIMET ADVANCED firmware will take into account only the first connection time. The logs' time, as well as log file time are specified in UTC time.
- If it is standalone without a CSM running, the timing starts at 0 when the ULTIMET ADVANCED is booted.
- If it is standalone with a CSM running, it depends on the CSM:
  - if the CSM uses its own TCPIP connection, then the user has the possibility to send it a time that can be transferred to the ULTIMET ADVANCED firmware via the *SLT=<Epoch time> command.

Command	***SLT = <P1>**
Description	Set Local Time.
Parameter	**Description**
<P1>	Epoch time (seconds from 1 jan 1970).

  - otherwise the timing will have started at 0 when the ULTIMET ADVANCED is booted.

9 Synchronized homing on several axes

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

Command	***IND = <P1>[,<P2>]**
Description	Starts a homing sequence on a group of axes
Parameter	**Description**
<P1>	Axes mask
<P2>	Disable adjustment movement (disable = 255)

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 ADVANCED 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 brake together and go in the opposite direction. This is to avoid 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 ADVANCED 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	
22									Immediate
38	+	X					OK		The trip is defined by KL46 after having found the mechanical end stop
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 §33. for more information these errors.

Example with a gantry system:

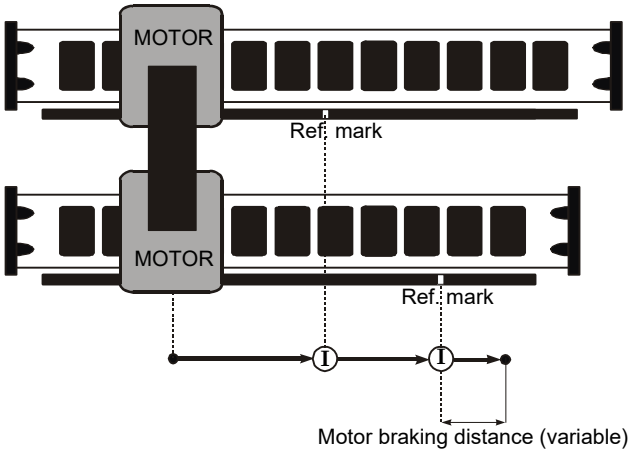
Here is a description of the symbols used in this paragraph:

Ⓘ Reference position
Reference mark
⓪ Zero machine
Motor position
Motor trip

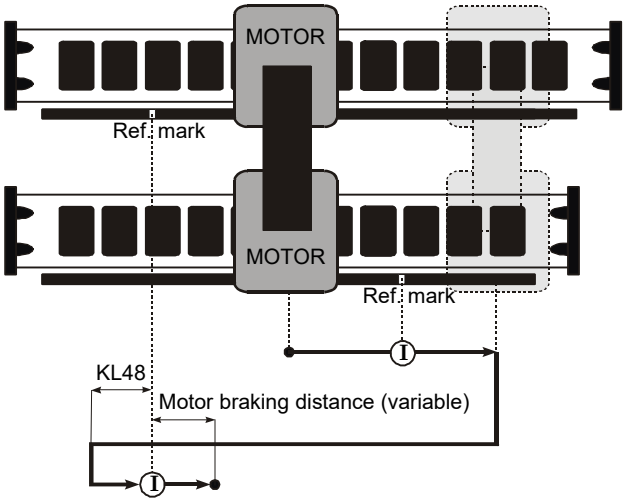
K40=8:

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:

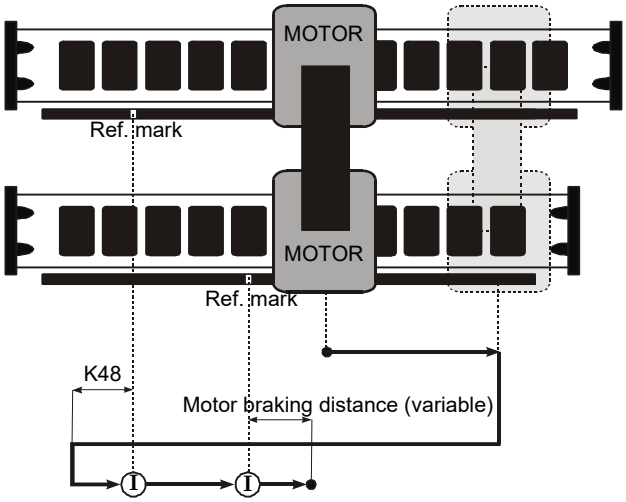
1. Both motors are on the left of the mono-reference marks at the beginning of the homing. They move and meet successively both reference marks.



2. 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.

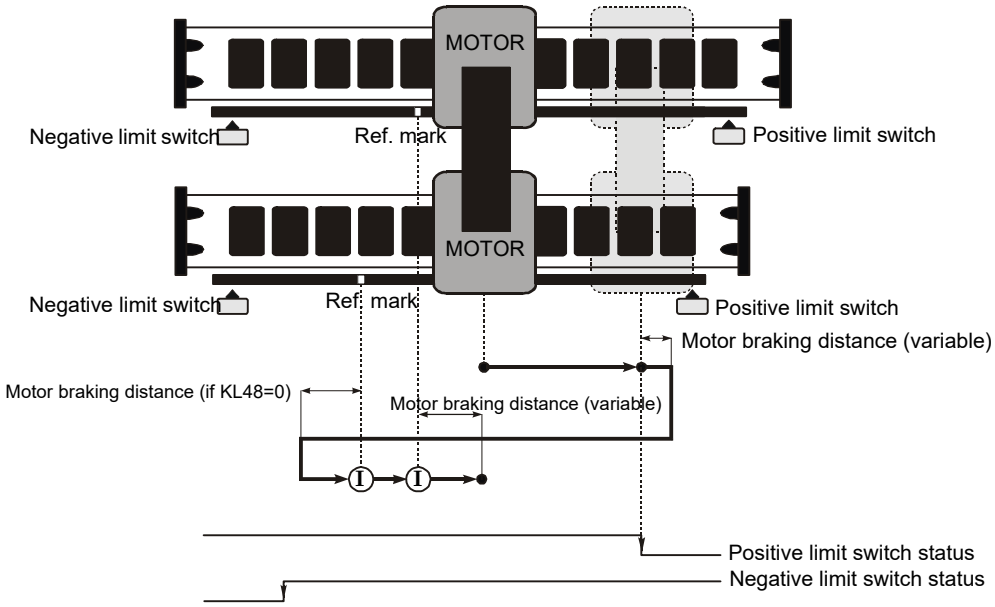


K40=9:

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.

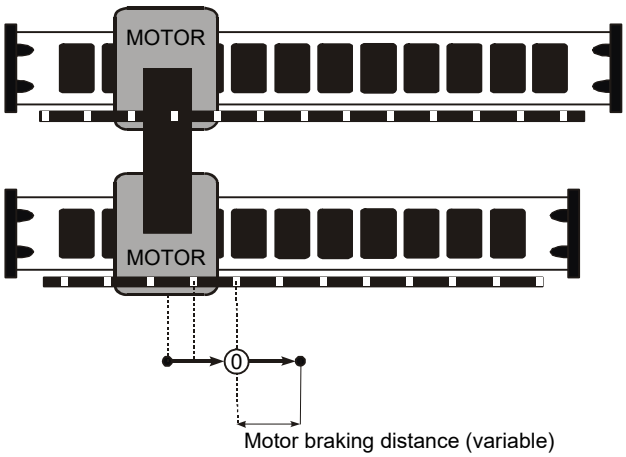


K40=11:

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.

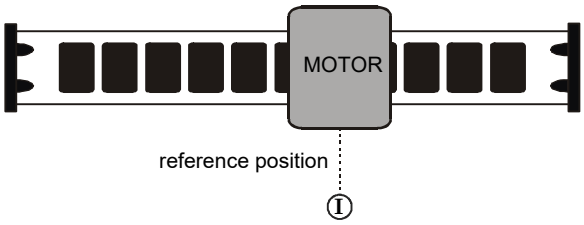


K40=13:

Same as K40=12 but with a negative movement.

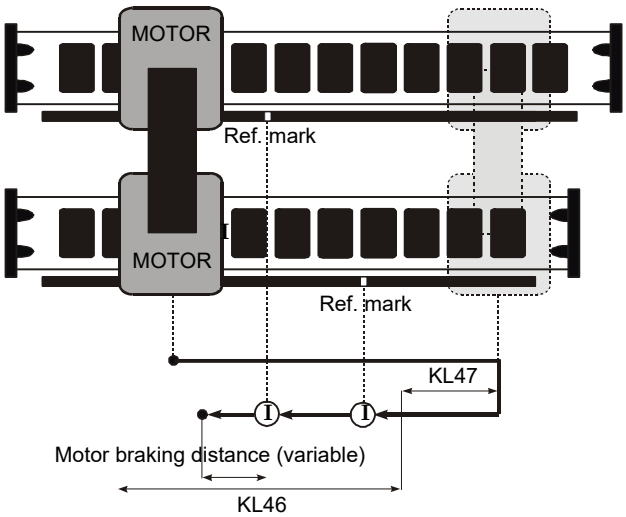
K40=22:

The present position is the reference position.



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.



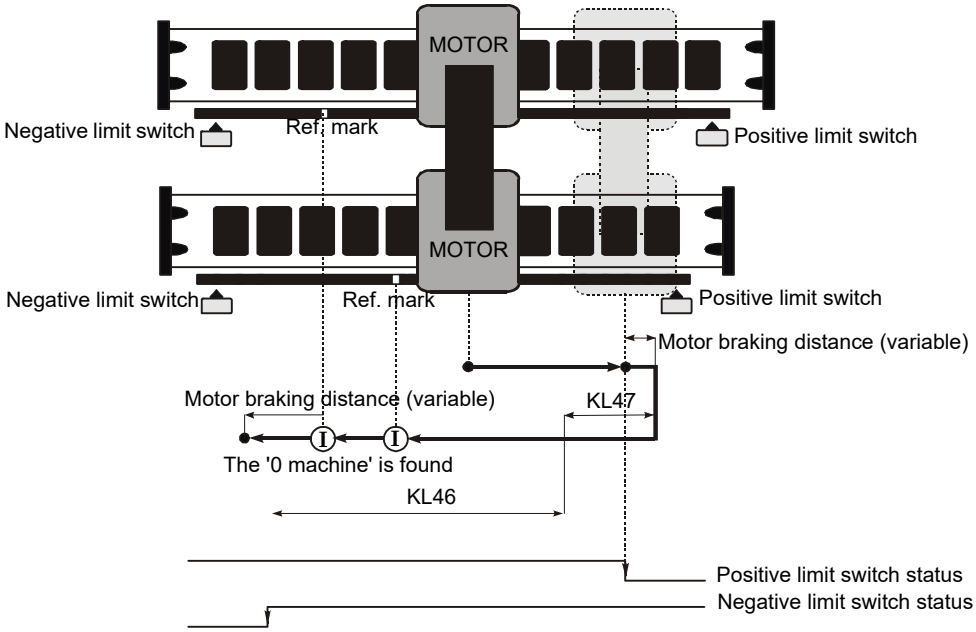
K40=39:

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.

10 Traces Management

The trace management allows the user to acquire up to 4 traces, 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 CSM.

Refer to the "EDI User's Manual" for the list of predefined functions available and how to use them. The description below is provided for reference only.

10.1 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)
***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)

K	Name	Comment
***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 µs.

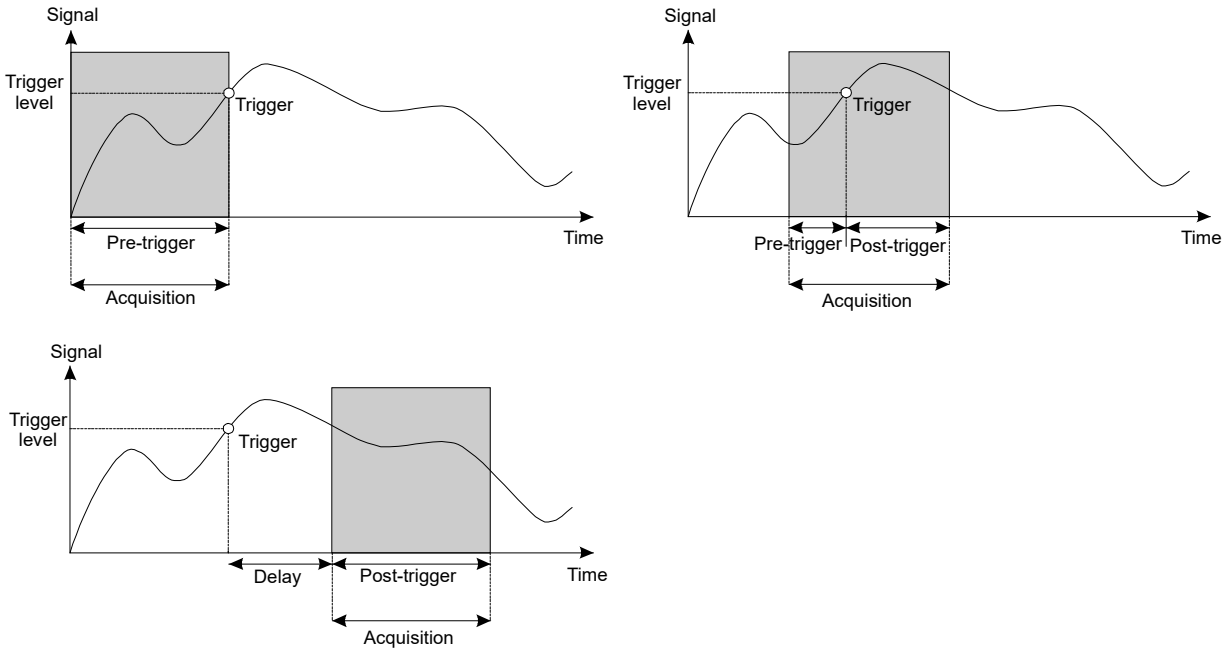
The traces configuration is stored into an internal structure and is accessible through monitoring registers. With these monitoring registers, the application (ComET, user application, sequence) has the information on what has been traced. Typically, ComET will be able to upload unknown traces of a controller 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** ***MD127** ***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

10.1.1 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, monitoring registers M135 and M136 have to be checked.

10.2 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	***ZTE[.<axis>]=<P1>**		
Description	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.		
Parameter	**Description**	**Value**	**Description**
<P1>	Enable state	0	Stops the acquisition (nevertheless the start and end index are available)
		1	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 §10.3)
		2	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 §10.3).

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 register ML7 and the real force given by the monitoring register MF31. The acquisition has 16001 points with a sampling time of 50us and lasts 800ms. The trigger is raising edge sensitive and is set on the real position (ML7) with a level of 1000000incr. A pre-trigger of 200ms 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;                  // Definition of the trigger level
KF127.0 = 0.0F, 0.0F;           // Definition of the trigger level
KL127.0 = 1000000L, 0L;         // Definition 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 M131.0 = 5 or when the bit#27 of M63.0 is reset, the acquisition is finished and the points can be read.

10.3 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 §10.2) 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 register **M228** or **ML228**.

M	Name	Comment
***M228** ***ML228**	Reference time counter	Reference time counter in unit of 25 µs. 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 register ML7 and the real force given by the monitoring register MF31. The reference time counter given by the monitoring register M228 is measured. The acquisition has 10001 points with a sampling time of 100 µs (the minimum possible with UltimET) and lasts 1000 ms. The trigger is raising edge sensitive and is set on the real position (ML7) with a level of 1000000 incr. A pre-trigger of 200 ms 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;                  // Definition of the trigger level
KF127.0 = 0.0F, 0.0F;           // Definition of the trigger level
KL127.0 = 1000000L, 0L;         // Definition 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;                   // Definition of the trigger level
*KF127 = 0.0F, 0.0F;            // Definition of the trigger level
*KL127 = 0L, 0L;                // Definition of the trigger level
*KD127 = 0.0D, 0.0D;            // Definition 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
```

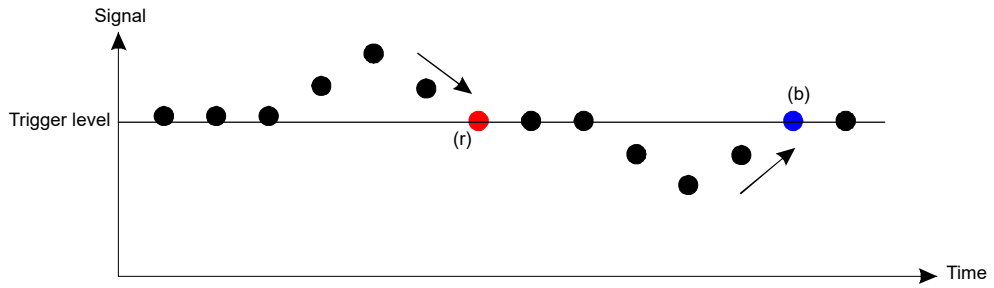
When *M131 = 5 or when the bit#27 of *M63 is reset, the acquisition is finished and the points can be read.

10.4 Trigger condition test and edge detection

The trigger condition is tested at each minimum sampling period (50 µs for AccurET and 100 µs 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 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.



10.5 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 register 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 §10.2 can be done by asking the value TL0.0, TL1.0, TL2.0, …, TL16000.0 to the controller.

10.6 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	***WTD [.<AXIS>] = <P1>, <P2>**
Description	WaiT for all Data are available for the Continuous Traces. Wait for the end of the acquisition of the points from <P1> to <P2>.
Parameter	**Description**
<P1>	First index for the range of data
<P2>	Last index for the range of data

In addition, a warning (bit#30 of M66) 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.

11 ULTIMET ADVANCED software characteristics registers

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

M	Alias	Value	Comment
***M70**	-	51 x	Depth 0:Indicates that the device is an ULTIMET ADVANCED Depth 2: Indicates the type of ESM – if any is loaded (0 if not).
***M71**	-	-	Gives the software boot version of ULTIMET ADVANCED.
***M72**	VER	-	Depth 0: Gives the firmware version of the ULTIMET ADVANCED Depth 2: gives the ESM version – if any is loaded (0 if not)
***M73**	SER	-	Gives the serial number of the ULTIMET ADVANCED
***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 ULTIMET ADVANCED version read in *VER (monitoring register *M72) is a hexadecimal number, with the following format:
*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	...	$ß_0$	$ß_1$	...	A	B	C	...	released	in dev.

For example, the hexadecimal value 0x**1208000** means that the firmware version is 1.20A.

12 ETEL Software Module

The ULTIMET ADVANCED firmware can host plugins that implement high-level ETEL proprietary features or algorithms, called ETEL Software Modules (ESM). They can be general purpose features, system specific features or customer specific requirements implemented by ETEL. For example, systems having a Moving Metrology Frame (MMF) require an ESM that implements the setup and computation of MMF data to control the system to its best accuracy. Also, high precision systems require that all vibrations or reaction movements be damped out. This would be provided by an ESM containing a so-called QUIET submodule.

ESMs use dedicated register types:
- W (write) registers for configuration, are equivalent to usual K registers.
- R (read) registers for monitoring, are equivalent to the usual M monitoring registers.

They come in the same flavours as the K and M registers, namely W, WF, WL, WD and R, RF, RL, RD.

There are registers and commands provided by and for all ESMs. These are described in this chapter. ESMs also bring with them their own specific commands and specific W and R registers. Please, refer to the ESM specific documentation to find out what commands and registers it provides.

If the customer application requires an ESM, it is bundled into the downloadable .far file provided by ETEL and installed by the ComET download tool on the ULTIMET ADVANCED.

12.1 Common registers

ESMs are made up of submodules that each implement different features. ESMs are provided for a given system platform and assembled with the submodules the platform is capable of providing.

At the time of this document release, the available ESM IDs are:

ESM ID	ESM name	Comment
2	QUIET	ESM handling a QUIET isolation system

The current submodule IDs are:

Submodule ID	Submodule name
0	Core
100	QUIET

***K15** allows choosing what submodule to activate or not:

K	Value	Comment
***K15**	0 Bitmask	No ESM is loaded (default value). Activate those submodules whose index bit is set.

The ULTIMET ADVANCED firmware loads and start the ESM automatically when *K15 is different from 0. The user needs to refer to the platform specific documentation to know the index of a submodule feature or refer to §12.2.

*K15 has to be set, saved and the ULTIMET ADVANCED rebooted so that the ESM is automatically loaded and started when the controller starts-up. The ESM itself then activates or not the submodules as specified in *K15.

The number of submodules present in an ESM is given by *M440:

M	Name	Comment
***M440**	Submodules count	Number of constituent submodules

ESMs are identified by an ID and a version number. They can be read from *M70:2 and *M72:2. They also have

a name that can be read out from *M84.

M	Name	Comment
***M70:2**	ESM ID	2: QUIET
***M72:2**	ESM version	
***M84**	ESM name	Gives the name of the ESM currently loaded.

You can also get each of the submodules' versions.

M	Name	Comment
***M441**	Submodules signature	
***M441:0**	ESM core submodule ID	
***M441:1**	ESM core submodule version	
***M441:2x**	Submodule ID	*M441:<2*(module_index+1)> = ID of submodule with index module_index
***M441:2x+1**	Submodule version	*M441:< 2*(module_index+1)+1> = version of module with index module_index

Each submodule also has a name that can be read in *M442 using the ***ESMSUBNAME** command.

Command	***ESUBNAME=<P1>**		
Description	Get the name of submodule indexed <P1>.		
Parameter	**Description**	**Value**	**Description**
<P1>	Submodule index	0 - 4	Returns the name of submodule <P1> in *M442

M	Name	Comment
***M442**	Submodule name	The name of the submodule whose index was specified in the *ESUBNAME command.

Example:

With an ESM that contains 2 submodules:

```
*M440 = 2 (number of submodules)
*M441:2 = ID of module indexed 0
*M441:3 = version of module indexed 0
*M441:4 = ID of module indexed 1
*M441:5 = version of module indexed 1
```

If there is only the QUIET submodule in the ESM:

```
*M440 = 1
*M441:2 = 100 (QUIET ESM submodule ID)
*M441:3 = 0x01008000 (in this case V1.00A)
*ESUBNAME=1 sets
*M442="QUIET"
```

Indicates that *MF320, *M321 and *MD310 are used by the ESM for reading RTV slots.

If you are writing an application that has to work with a given submodule contained in any ESM you will probably need to find out the index of that submodule:

- Either you hard code the index in function of the ESM's name or Id having referred to the ESMs documentation to find out its submodules' indices.
- Or, a more generic method is to loop with an index ranging from 0 to *M440 and check if *ESMSUBNAME=index returns in *M442 the name of the submodule being looked for.

Each submodule also has a status word:

M	Value	Comment
***M439:<x>**		Submodule status. Status word of submodule with depth index <x>.
	Bit 0	Initialized
	Bit 1	Enabled

M	Value	Comment
	Bit 2	Activated
	Bit 3	High priority begin task running
	Bit 4	High priority end task running
	Other bits	Submodule dependent

12.2 RTV consideration

The ESM runs as part of the firmware, and they share some of the resources. It may also use resources otherwise made available to a customer application like RTV slots. The firmware monitoring registers *M448 and *M449 inform the user which slots are used for writing. Refer to §14 for their detailed description.

When an ESM uses RTV monitoring registers for reading, the user application must not assign those monitoring registers for its own use. This is why two monitoring registers are available to inform a customer application which RTV monitoring registers are already used by the ESM:

M	Comment
***M444**	The number of RTV monitoring registers used by the current ESM
***M445**	This monitoring register has twice as many depths as there are slots used for reading: - Even depths give the index of the RTV read monitoring register - Odd depths gives the data type of the monitoring register: 0: 32-bit integer 1: 32-bit float 2: 64-bit integer 3: 64-bit double

Example:

```
*M444 = 3
*M445:0 = 320
*M445:1 = 1
*M445:2 = 321
*M445:3 = 0
*M445:4 = 310
*M445:5 = 3
```

Indicates that *MF320, *M321 and *MD310 are used by the ESM for reading RTV slots.

12.2.1 Commands

In addition to the ***ESMSUBNAME** command, three other common commands are available to control an ESM:

Command	***ESMSUBEN=<P1>,<P2>**		
Description	Performs the “dynamic” initializations like RTV setup of submodule indexed <P1>. This must be the first step of ESM enabling.		
Parameter	**Description**	**Value**	**Description**
<P1>	Submodule index	0 - 4	
<P2>	On/Off	0 1	Disable Enable

Caution: This command typically allocates or release RTV monitorings and slots. Therefore, use any command or EDI function that resets RTVs with caution and in proper order with respect to this command:

- Do not reset all RTVs after having called *ESMSUBEN=<index>,1. Otherwise the slots setup by the ESM will no longer work.
- Do not call *ESMSUBEN=<index>,0 after having reset all RTVs as that will create an ESM error because it will try to free RTVs no longer assigned.

The correct order of operations is:

- Reset all RTVs if needed

- *ESMSUBEN=<index>,1
- … (in operation)
- *ESMSUBEN=<index>,0
- Reset all RTVs if needed

When the *ESMSUBEN has been executed and the rest of the customer application is initialized, the ***ESMSUBACT** command must sent to start the submodule operations.

Command	***ESMSUBACT=<P1>,<P2>**		
Description	Switches submodule indexed <P1> into full operation, typically starting high priority activities, and required power on… This must be the last step of ESM enabling.		
Parameter	**Description**	**Value**	**Description**
<P1>	Submodule index	0 - 4	
<P2>	On/Off	0 1	Disable Enable

If an AccurET RTV monitoring registers must be used by the application for specific purposes, the ***ESMRTVRSVD** must be used to notify the ESM that it must not use that monitoring for itself.

Command	***ESMRTVRSVD=<P1>,<P2>,<P3>**		
Description	Indicates to the ESM core RTV manager that register <P1> on axis <P2> is not to be used in the automatic RTV monitoring assignment, typically because it is required to be reserved by the application.		
Parameter	**Description**	**Value**	**Description**
<P1>	Register specification		
<P2>	Axis number		
<P3>	Reserved / unreserved	0 1	Unreserve Reserve

12.3 ESM error codes

The common ESM error codes can be found in §33.2. Submodule specific error codes are to be found in their respective documentation.

13 Local digital Inputs / Outputs

General purpose I/Os are available on the ULTIMET ADVANCED on connectors X5 and X6 (refer to the corresponding “Hardware Manual” for more information). There are 4 I/Os on each connector. Each I/O can be configured either as digital input or digital output or to be used for the SPI feature (refer to §16 for more information). The “Hardware Manual” describes how the I/Os are numbered with respect to the connector pins. The I/Os are referred to by a bit mask:

I/O #	8	7	6	5	4	3	2	1
Bit #	7	6	5	4	3	2	1	0
Hexadecimal value	0x80	0x40	0x20	0x10	0x8	0x4	0x2	0x1

13.1 Digital inputs

The digital inputs can be used either as simple direct input or to trigger an ULTIMET ADVANCED error as a reaction to a protection signal. The monitoring register ***M806** gives the mask of all the I/Os configured as inputs, irrespective of their usage.

M	Name	Comment
***M806**	I/Os as inputs	Gives the mask of the digital I/Os configured as direct inputs

Example:

```
*M806 = 0x6
```

Says that I/O #2 and #3 are used as inputs. One may be a direct input, and the other used for the protection signal.

The specific usage of inputs is described below.

13.1.1 Direct digital input

13.1.1.1 Configuration

The parameter ***K804** specifies the mask of I/Os used for direct input.

K	Name	Comment
***K804**	Direct input I/O mask	Bit mask of the general I/Os to be set as direct input

The monitoring register ***M804** gives the mask of the I/Os configured as direct input.

M	Name	Comment
***M804**	I/Os as direct inputs	Gives the mask of the digital I/Os actually configured as direct inputs

*M804 and *K804 may differ in case the value specified in *K804 addresses I/Os already configured for another purpose (as output for a protection signal, e.g.). Refer also to §13.1.3.

13.1.1.2 Register Access

The monitoring register ***M50** allows the user to read the actual state of all the digital inputs. The ***DIN** command is an alias of this monitoring register and uses the same syntax.

M	Alias	Name	Comment
***M50**	*DIN	DIN states	Gives the states of ALL the digital inputs

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 the “Hardware Manual” for more information.

Example:

If `*K804 = 0x30` and `*M804 = 0x30` as well, the configuration is OK: I/Os #4 and #5 as direct input. `*M50 = 0x10` means digital input #4 is high and digital input #5 is low. Nothing else can be said of the other I/Os (not configured as inputs).

13.1.1.3 Command Access

There are two commands to access the direct digital inputs, each having one form to be used from the ComET terminal, and one from an EDI (PC or CSM) application.

To get the value of a specific digital input:

- From the ComET Terminal:

Command	***GETDINS = <P1>, <P2>**
Description	Get the actual value of a specified digital local input. Call with P2=0. When command returns, P2 = value of the input (0 or 1).
Parameter	**Description**
<P1>	Index of the digital local input to get
<P2>	Value of the input

- From an EDI application:

```
int dsa_localIO_get_digital_input_state_s(DSA_DEVICE *device,
                                          int io_number,
                                          dword *value,
                                          ebool fast,
                                          long timeout);
```

The value of the digital input (0 or 1) is placed in `value`. Refer to EDI User's Manual for information about all the other parameters.

To get the value of a set of inputs:

- From the ComET Terminal:

Command	***GETMDINS = <P1>, <P2>, <P3>**
Description	Get the state of a set (mask) of local digital inputs. Call with P3=0. When command returns, P3 = the state of the specified number of inputs starting from the specified index.
Parameter	**Description**
<P1>	Index of the first input to get
<P2>	Number of inputs to get
<P3>	Value of the masked input

- From an EDI application:

```
int dsa_localIO_get_masked_digital_input_state_s(DSA_DEVICE *device,
                                                 int first_input_idx,     // P1
                                                 int number_of_inputs,    // P2
                                                 int *value,              // P3
                                                 bool fast,
                                                 long timeout)
```

All `number_of_inputs` I/Os starting from `first_input_idx` must be configured as direct digital inputs. The state of the inputs are returned as a mask in value with the LSB corresponding to `first_input_idx`.

Example:

Consider the configuration is `*K804 = 0xF` – first 4 I/O pins configured as direct input. Inputs 2 and 4 are high – i.e. *DIN, expressed in binary format, yields 1010.
Calling `*GETMDINS = 1, 3, 0` returns `*GETMDINS = 1, 3, 5` – i.e the return digital inputs mask in binary is 101.

Similarly, calling `dsa_localIO_get_masked_digital_input_state_s(dev, 1, 3, &value, false, DEFAULT_TIMEOUT)` returns with value equal to 5 (binary 101).

Either of these forms of the command can raise the error `*M64 = 120130` if one of the `number_of_inputs` starting from `first_input_idx` is not configured as a direct digital input.

13.1.2 Protection signals on digital inputs

The parameter ***K305** specifies which inputs are used to generate an error on the ULTIMET ADVANCED.

K	Name	Comment
***K305**	Protection signal input I/O mask	Bit mask of the input pins to be used to generate an error linked to a protection signal: Bits 0-7: I/Os that generate an error when their level is high, Bits 16-23: I/Os that generate an error when their level is low.

The monitoring register ***M305** gives the mask of the I/Os actually configured as protection input.

M	Name	Comment
***M305**	I/Os as direct inputs	Gives the mask of the digital I/Os actually configured as protection signal

*M305 and *K305 may differ in case the value specified in *K305 addresses I/Os already configured for another purpose (as output, e.g.) or specifies an input set to react both on high and low levels. Refer also to §13.2.5.

To enable this feature, *K33 must be set accordingly. It is bit#4 of this parameter that allows the user to raise 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 states of the digital inputs matches with the mask *K305.

When the I/Os specified in *K305 match the levels also specified in *K305, the following error is generated: *M64=4051.

13.1.3 Troubleshooting

The following errors can occur when configuring the digital I/Os for some input usage:

Error returned in *M64	Meaning
4503	Configuring one of the I/O pins specified in either *K804 or *K305 as input failed because it is already configured for another non digital I/O function like SPI.
4508	In *K305, an I/O was specified both in bit range 0-7 and range 16-23, meaning both high and low levels of this IO triggers an error on the ULTIMET ADVANCED.
4500	Some of the I/Os referenced in either *K804 or *K305 are already configured for another digital I/O purpose (output, input but other feature).
4513	An attempt has been made (either via a command or an EDI call) to get the value of a digital input with an I/O number of an I/O not configured for direct input.

Recovery: Review the parameters settings (including the output configuration parameters) to resolve the conflicts.
You can also completely reset the digital IO configuration with the **IOCFGRST**. To completely reset all I/O pins usage you need to reset the SPI configuration, if it is used, using the command **SPIRST**. And then, start the configuration over again.

Command	***IOCFGRST**
Description	Resets IO pins configuration to "none used"

Command	***SPIRST**
Description	Reset SPI configuration.

13.2 Digital outputs

The digital outputs can be used for several features:
- Direct output
- To output a signal when the ULTIMET ADVANCED generates an error
- To output a signal with a frequency multiple of and synchronized with the TransnET cycle
- For the interpolation's use (refer to the interpolation section).

The monitoring register *M807 gives the mask of all the I/Os configured as outputs, irrespective of their usage.

M	Name	Comment
***M807**	All I/Os as outputs	Gives the mask of all the digital I/Os actually configured as outputs.

13.2.1 Direct digital output

13.2.1.1 Configuration

The parameter ***K805** specifies the mask of I/Os used for direct output.

K	Name	Comment
***K805**	Direct output I/O mask	Bit mask of the general I/Os to be set as direct output

The monitoring register ***M805** gives the mask of the I/Os configured as direct output.

M	Name	Comment
***M805**	I/Os as direct outputs	Gives the mask of the digital I/Os actually configured as direct outputs

*M805 and *K805 may differ in case the value specified in *K805 addresses I/Os already configured for another purpose (e.g. as output of TransnET signal). Refer also to §13.2.5.

13.2.1.2 Register Access

The parameter ***K171** allows the user to activate or deactivate the digital outputs. The *DOUT command is an alias of this parameter and uses the same syntax.

K	Alias	Name	Comment
***K171**	*DOUT	DOUT values mask	Activates / deactivates the digital outputs.

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 the "Hardware Manual" for more information.

The actual state of the digital outputs can be read with the monitoring register ***M171**:

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

Example:

If `*K805 = 0xC` and `*M805 = 0xC` as well, the configuration is OK: I/O #2 and #3 as direct output.
Setting `*K171 = 0x4` will set I/O #2 high and I/O #3 low. *M171 will read `*M171 = 0x4`.

Attempting to set `*K171 = 0x1` will raise an error (I/O #0 not configured as output). Refer also to §13.2.5.
*M171 will remain at `*M171 = 0x4` (previous value).

13.2.1.3 Command access

There are a couple of commands to manipulate the direct outputs using commands either from the ComET terminal or from an EDI application.

To get a specific direct output value:

- From the ComET terminal:

Command	*GETDOUT = <P1>, <P2>
Description	Return the state (0 or 1) of the specified local output.
Parameter	**Description**
<P1>	Index of the digital local output to get
<P2>	Value of the output

- From an EDI application:

```
int dsa_localIO_get_digital_output_s(DSA_DEVICE *dev,
                                     int output_idx,      // P1
                                     dword *value,        // P2
                                     ebool fast,
                                     long timeout);
```

To set and reset a specific digital output:

- From the ComET terminal:

Command	*SETDOUT = <P1>
Description	Set a specified digital local output
Parameter	**Description**
<P1>	Index of the digital local output to set (value = 1)

Command	*RSTDOUT = <P1>
Description	Reset a specified digital local output
Parameter	**Description**
<P1>	Index of the digital local output to reset (value = 0)

- From an EDI application:

```
int dsa_localIO_set_digital_output_s (DSA_DEVICE *dev,
                                      int output_idx,      // P1
                                      long timeout);

int dsa_localIO_reset_digital_output_s (DSA_DEVICE *dev,
                                        int output_idx,    // P1
                                        long timeout);
```

To get the value of a set of outputs:

- From the ComET Terminal:

Command	*GETMDOUT = <P1>, <P2>, <P3>
Description	Get the state of a set (mask) of local digital outputs Call with P3=0. When command returns, P3 = the state of the specified number of outputs starting from the specified index.
Parameter	**Description**
<P1>	First output bit of the mask
<P2>	Number of bits of the mask
<P3>	Value of the masked output

- From an EDI application:

```
int dsa_localIO_get_masked_digital_output_state_s(DSA_DEVICE *device,
                                                  int first_output_idx,    // P1
                                                  int number_of_outputs,   // P2
                                                  int *value,              // P3
                                                  bool fast,
                                                  long timeout)
```

All `number_of_outputs` I/Os starting from `first_output_idx` must be configured as direct digital outputs. The state of the outputs are returned as a mask in `value` with the LSB corresponding to `first_output_idx`.

Example:

Consider the configuration is `*K805 = 0xF0` – last 4 I/O pins configured as direct output. Outputs 6 and 8 are high – i.e. *DOUT, expressed in binary format, yields 1010 0000.

Calling `*GETMDOUT = 5, 3, 0` returns `*GETMDOUT = 5, 3, 5` – i.e. the return digital outputs mask in binary is 101. Similarly, calling `dsa_localIO_get_masked_digital_output_state_s(dev, 5, 3, &value, false, DEFAULT_TIMEOUT)` returns with value equal to 5 (binary 101).

Either of these forms of the command can raise the error `*M64 = 120140` if one of the `number_of_outputs` starting from `first_output_idx` is not configured as a direct digital output.

To set the value of a set of outputs:

- From the ComET Terminal:

Command	***SETMDOUT = <P1>, <P2>, <P3>**
Description	Apply a mask on local digital outputs
Parameter	**Description**
<P1>	First output bit of the mask
<P2>	Number of output bits of the mask
<P3>	Value to put on masked digital output

- From an EDI application:

```
int dsa_localIO_apply_mask_digital_output_s (DSA_DEVICE *device,
                                             int first_output_idx,    // P1
                                             int number_of_outputs,   // P2
                                             int *value,              // P3
                                             long timeout)
```

All number_of_outputs I/Os starting from first_output_idx must be configured as direct digital outputs.

13.2.2 Protection signals on digital output

When a motor is integrated in a complex machine and an error is detected in the ULTIMET ADVANCED, it is important to send a message about it to the rest of the machine so that the other elements can react accordingly. This is possible with the digital outputs.

The parameter ***K357** allows the user to set or reset 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 digital outputs if error	Mask of the digital outputs that must be set or reset when the ULTIMET ADVANCED is in error. Bits 0-15: set, bits 16-31: reset.

The monitoring register ***M809** gives the mask of the I/Os actually configured as protection output.

M	Name	Comment
***M809**	I/Os as output protection signal	Gives the mask of the digital I/Os actually configured as output protection signal.

*M809 and *K357 may differ in case the value specified in *K357 addresses I/Os already configured for another purpose (e.g. as direct output) or specifies an output to be set both high and low on an ULTIMET ADVANCED error. Refer also to §13.2.5.

13.2.3 TransnET signal output

On ULTIMET ADVANCED it is possible to output the TransnET signal on an I/O, or a synchronized sub-frequency of it. The parameter ***K353** allows the user to select the I/Os on which to output the TransnET signal.

K	Name	Comment
***K353**	I/Os as TransnET output	Mask of the I/Os outputting the TransnET signal.

The monitoring register ***M808** gives the mask of the I/Os actually configured as protection output.

M	Name	Comment
***M808**	I/Os as TransnET output	Mask of the I/Os actually outputting the TransnET signal.

*M808 and *K353 may differ in case the value specified in *K353 addresses I/Os already configured for another purpose (e.g. as direct output). Refer also to §13.2.5. The sub-frequency of the TransnET cycle can be selected by setting the desired multiple of the TransnET period using parameter ***K360**.

K	Name	Comment
***K360**	Multiple of the TransnET period	Multiple of the TransnET period.

Example:

`*K353 = 0x1` and `*K360 = 2` will output a signal synchronized with the TransnET cycle at half the frequency on digital output 1. The monitoring register *M360 reflects the value set by *K360.

13.2.4 Interpolation digital output

Some I/Os can be reserved for the interpolation to be set upon interpolation events. Refer to §15.6.2 for a description on how the interpolation uses digital outputs. The parameter ***K541** is provided to this effect.

K	Name	Comment
***K541**	I/Os mask	I/O reservation for interpolation.

The monitoring register ***M541** gives the mask of the I/Os configured as direct output.

M	Name	Comment
***M541**	Interpolation reserved IOs	I/Os actually reserved for interpolation.

*M541 and *K541 may differ in case the value specified in *K541 addresses I/Os already configured for another purpose (e.g. as direct input). Refer also to §13.2.5.

13.2.5 Troubleshooting

The following errors can occur when configuring the digital I/Os for some output usage:

Error returned in *M64	Meaning
4504	Configuring one of the I/O pins specified in either *K805, *K353,* K357 or *K541 as output failed because it is already configured for another non digital I/O function like SPI.
4506	Attempt to set an output (*K171) not configured as a direct digital output.
4510	In *K357, an I/O was specified both in bit range 0-7 and range 16-23, meaning an error on the ULTIMET ADVANCED would try to set an output both high and low.
4500	Some of the I/Os referenced in either *K805, *K353,* K357 or *K541 are already configured for another digital I/O purpose (input or output, but of other feature).
4516	The interpolation feature (refer to §13.2.4) has attempted to set a digital output that has not been reserved for its usage in *K541.
4517	Erroneous attempt to set a digital output for the TransnET signal with a value of *K353 that specifies a pin already assigned to another function.
4518	The value of *K360 (output TransnET period multiple) outside of range (0 to 65535).

14 Real-time channels on TransnET

14.1 Principle

In some applications it is necessary to transmit data between the ETEL controllers. There are two ways to share the data between the AccurETs, the UltimET and the PC:

- By sending commands from an ULTIMET ADVANCED CSM 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 bus using specific commands that are going to be introduced in this chapter.

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

- The UltimET and AccurET controllers.
- The AccurET controllers themselves.
- The application PC and the controllers (with ULTIMET ADVANCED through PCIe application bus only).
- The CSM and the controllers.

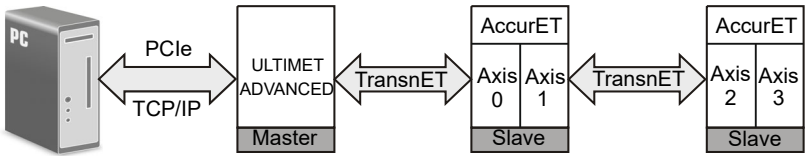
The TransnET bus works with cycle times of 100 µs. Each 100 µs, a TransnET frame starts from the ULTIMET ADVANCED (TransnET bus master) and is shared between the different connected position controllers. A part of the TransnET frame is made up of slots (a slot is a 32-bit data word) used for cyclic data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

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

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

The number of RTV slots depends on the number of axes present on the TransnET bus: 298 (maximum available) slots – (4 x number of axes (for monitoring request)) - (2 x number of interpolated axes).

Example:



The number of RTV slot is:

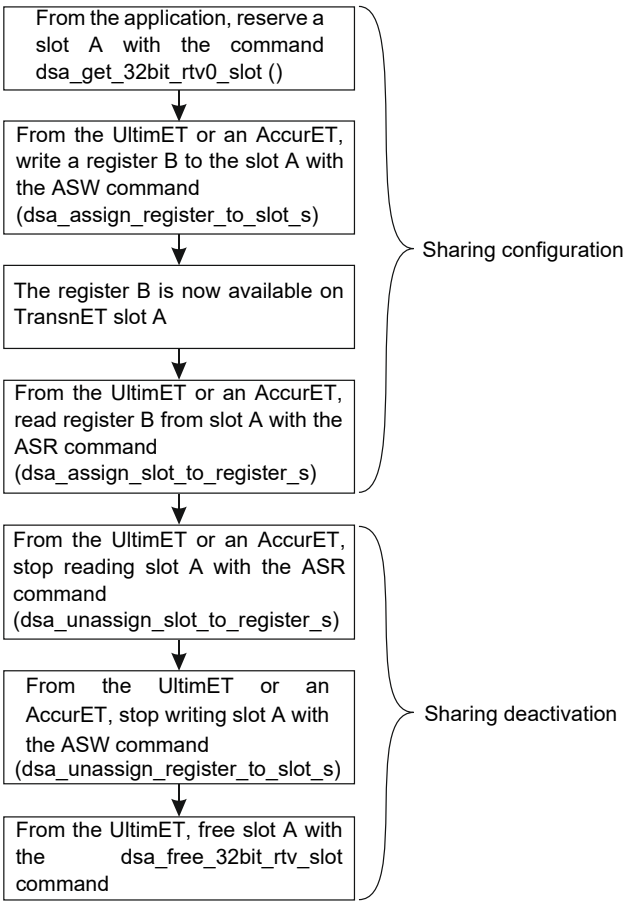
- Without interpolation: 298 - (4 x 4) = 282.
- With interpolation on axes 2 and 3: 298 - (4 x 4) - (2 x 2) = 278.

14.2 Data sharing

14.2.1 Data sharing between controllers

The read or write on these cyclic data slots is done through data sharing functions (refer to §14.4). The user has to ask the UltimET for free RTV slots before using these data sharing functions. These kind of functions allows one controller to have access to another controller's registers in real-time.

Here is a summary of the data sharing principle:



The number of 32-bit data slots that can be shared is:

	Read from TransnET	Write on TransnET
AccurET	8 or 16	7 or 15
ULTIMET ADVANCED	64	32
PC	All available slots (PC dependent)	All available slots (PC dependent)

The EDI interface provides access to all these functions. The request for RTV slots has to be done either on the PC application or from an ULTIMET ADVANCED CSM.

UltimET functions related to RTV are described here below, as well as most of EDI related functions. 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.

14.2.2 Data sharing between controllers and a PC application (PCIe only) or a CSM

When using real-time operating systems or using a CSM, a customer application has the possibility to have a direct access to the TransnET cyclic data slots at the PCIe interface or at the CSM interface.

EDI offers a set of functions allowing the sharing of real-time cyclic data with the controllers. A user defined handler can be called at each TransnET cycle (or multiples thereof) to execute cyclic tasks. A watchdog functionality is provided to warn the user if the execution of the user handler cannot be processed at the correct frequency (e.g. too much processing load in the handler).

14.2.3 Disappearance of TransnET

Each time a TransnET connection is established, the configuration of the TransnET slots is initialized. Depending on the number of axes, the RTV 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 PC application and/or in the ULTIMET ADVANCED CSM.

To notify this, bit#30 of the monitoring register *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 PC application level, and/ or in the CSM running on the ULTIMET ADVANCED.

14.3 Slot management functions

The following EDI functions are used to manage the slots in the ULTIMET ADVANCED.

Remark: The following functions cannot be executed from the 'Terminal' tool of ComET.

Function	Description
`int dsa_get_32bit_rtv0_slot( DSA_MASTER *dsm, DSA_RTV_SLOT **slot);` `int dsa_get_64bit_rtv0_slot( DSA_MASTER *dsm, DSA_RTV_SLOT **slot);`	Get a free TransnET slot (or two, if 64bit) parameter 1: master device address parameter 2: address of the `DSA_RTV_SLOT` pointer returns 0, if successful, a DSA error otherwise
`int dsa_free_32bit_rtv_slot( DSA_RTV_SLOT **slot);` `int dsa_free_64bit_rtv_slot( DSA_RTV_SLOT **slot);`	Free the specified `DSA_RTV_SLOT` pointer

The following UltimET monitoring registers are used to manage the slots:

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

Each time the 'dsa_get_<xx>bit_rtv0_slot()' command is executed, a new slot (or two) 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 ADVANCED and one AccurET
// controller (axes 0 and 1). The initial number of RTV slots is: *M524 = 298 -
// (2 x 4) = 290 free RTV slots.

DSA_RTV_SLOT* slot_axis0;       // Variable for the allocated slot number
int err = 0;

err = dsa_get_32bit_rtv0_slot(master, &slot_axis0); // *M524 is now equal to 289
err = dsa_get_32bit_rtv0_slot(master, &slot_axis0);
                 // Error: this second request of slot allocation must not be
                 // done using the same variable. *M524 is now equal to 288.

// At this point, the first slot number previously stored in slot_axis0 is no
// longer available.
dsa_free_32bit_rtv_slot(&slot_axis0);   // One slot, whose value is stored in
                                        // slot_axis0, is freed.
                                        // *M524 is now equal to 289.

// The first requested slot cannot be freed with the 'dsa_free_32bit_rtv_slot'
// command. There are two possibilities to free all the RTV slots:
// - Making 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, using the following
//   EDI function:
int dsa_free_all_transnET_slots_s(DSA_MASTER *master, long timeout);
```

14.4 UltimET – AccurET data sharing functions

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 RTV slot to the UltimET is mandatory.

14.4.1 ASW (ASsign for Write) command

Command	***ASW[.<axis>] = <P1>, <P2>, <P3>**
Description	Sets the slot (offset in the TransnET cyclic data) where the controller writes cyclic data RTV
Parameter	**Description**
<P1>	Register_to_share: defines the register to be written in the TransnET cyclic data (RTV). If the value=0, the register is not written anymore in the TransnET cyclic data.
<P2>	Slot_lsl: Defines the offset in the TransnET cyclic data where the controller writes the register lower part (RTV).
<P3>	[Slot_msl]: Defines the slot in the TransnET cyclic data where the controller writes the register, MSL part. Only available for 64-bit registers (RTV).

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

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

The number of possible RTV write slots is present in the monitoring register ***M443**.

M	Name	Comment
***M443**	Number of slots available for ASW	Number of possible RTV write slots (ASW)

The number of slots already used by the *ASW command can be obtained from monitoring register ***M449**. There are 32 slots reserved for this function for the ULTIMET ADVANCED. Each slot number associated with an ASW is accessible in ***M448**. The monitoring register depth corresponds to each of the 32 possibly reserved slots.

M	Name	Comment
***M449**	Counter of written RTV	Gives the number of written RTV (by *ASW command)
***M448**	Used RTV slots (by ASW)	Gives the slot number associated to a register (the depth corresponds to each of the 32 possibly reserved slots)

If the *ASW command does not have the right format (incorrect number of arguments, slot number out of range, etc.), the following errors can occur:

Error returned in M64	Meaning
58	'**BAD SLOT TRANSNET**' for AccurET
1053	'**Syntax error in internal command**' for UltimET
1210	'**ASW Slot not reserved**' for UltimET
1211	'**Not enough ASW slot available**' for UltimET
1221	'**Register Type does not fit with number of specified slot**' for UltimET
1222	'**Bad Slot Number specified**' for UltimET
1223	**'First Parameter should be a register address or 0'** for UltimET

EDI functions which execute this ASW command are:
- Assignment (synchronous and asynchronous)

```
int dsa_assign_register_to_slot_s(DSA_DEVICE *dev, int typ, int idx, int sidx,
                                  DSA_RTV_SLOT *slot, int timeout);

int dsa_assign_register_to_slot_a(DSA_DEVICE *dev, int typ, int idx, int sidx,
                                  DSA_RTV_SLOT *slot, DSA_HANDLER handler,
                                  void *param);
```

- Unassignment (synchronous and asynchronous)

```
int dsa_unassign_register_to_slot_s(DSA_DEVICE *dev, int typ, int idx, int sidx,
                                    DSA_RTV_SLOT *slot, int timeout);

int dsa_unassign_register_to_slot_a(DSA_DEVICE *dev, int typ, int idx, int sidx,
                                    DSA_RTV_SLOT *slot, DSA_HANDLER handler,
                                    void *param);
```

14.4.2 ASR (ASsign for Read) command

Command	***ASR[.<axis>] = <P1>, <P2>, <P3>**
Description	Gives the slot offset in the TransnET cyclic data from which to read the RTV data. <P1> is the Mx310 to Mx363, to be assigned with the data, <P2> is the slot offset from where it reads the low part and <P3> the high part in case of 64 bits.
Parameter	**Description**
<P1>	Monitoring_register: defines the monitoring register that is updated with the TransnET cyclic data slot value: M310 to M373, or MF310 to MF373, or ML310 to ML341 or MD310 to MD341.
<P2>	Slot_lsl: defines the slot number where the data or the lsl part of the data is found in the TransnET frame.
<P3>	[Slot_msl]: defines the slot number where, only for 64-bit data, the msl part is found in the TransnET frame.

To stop reading a register, the following command must be sent: ASR = monitoring_register, -1, [-1].

Remark: ULTIMET ADVANCED monitoring registers' range (from M310 to M373) is different from range of AccurET or UltimET Light (from M450 to M465).

The maximum number of possible RTV read slots is present in ***M447**.

M	Name	Comment
***M447**	Number of slots available for ASR	Maximum number of possible RTV read slots (ASR)

If the *ASR command does not have the right format (incorrect number of arguments, slot number out of range...), the following errors can occur:

Error returned in M64	Meaning
58	'**BAD SLOT TRANSNET**' for AccurET
1053	'**Syntax error in internal command**' for UltimET
1220	'**Not enough ASR slot available**' for UltimET
1223	'**First Parameter should be a register address or 0**' for UltimET
1224	'**First Parameter should be a M register address**' for UltimET
1225	'**ASR Slot not reserved**' for UltimET

EDI functions which execute this ASR command are:

- Assignment (synchronous and asynchronous)

```
int dsa_assign_slot_to_register_s(DSA_DEVICE_BASE *grp, DSA_RTV_SLOT *slot,
                                  int typ, int idx, int sidx, int timeout);

int dsa_assign_slot_to_register_a(DSA_DEVICE_BASE *grp, DSA_RTV_SLOT *slot,
                                  int typ, int idx, int sidx,
                                  DSA_HANDLER handler, void *param);
```

- Assignment (synchronous and asynchronous)

```
int dsa_unassign_slot_to_register_s(DSA_DEVICE_BASE *grp, DSA_RTV_SLOT *slot,
                                    int typ, int idx, int sidx, int timeout);

int dsa_unassign_slot_to_register_a(DSA_DEVICE_BASE *grp, DSA_RTV_SLOT *slot,
                                    int typ, int idx, int sidx,
                                    DSA_HANDLER handler, void *param);
```

Example:

```
// Sharing enabling/disabling of the real position of the axis 0 to axes 1, 2 and
// 3 and UltimET.

#define MASTER_RTV_MON_REG_IDX 310
#define SLAVES_RTV_MON_REG_IDX 450

// Consider:
// - A device group exists, made of axes 1, 2 and 3: grp
// - A master device exists, made of ULTIMET ADVANCED: master
// - A drive device exists, made of axis 0: axis0

DSA_RTV_SLOT* axis0_real_position_slot;  // Pointer on the RTV slot with
                                         // the real position ML1

// Next function shares real position ML1 of axis 0, and axes 1, 2, 3
// and the UltimET are set up to read the shared position.
int SharePositionAxis0(void)
{
    int err = 0;

    err = dsa_get_64bit_rtv0_slot(master, &axis0_real_position_slot);
    if (err) {
        DSA_EXT_DIAG(err, master);
        return err;
    }

    // Axis 0 shares its real position ML1 on TransnET, using two 32-bit slots
    err = dsa_assign_register_to_slot_s(axis0, DMD_TYP_MONITOR_INT64,
                                        DMD_ML_POSITION_ACTUAL_VALUE_DS, 0,
                                        axis0_real_position_slot,
                                        DSA_DEF_TIMEOUT);
    if (err) {
        DSA_EXT_DIAG(err, master);
        return err;
    }

    // Axes 1, 2, 3 and UltimET can access the axis 0 real position through
    // their own ML450 monitoring register
    err = dsa_assign_slot_to_register_s(master, axis0_real_position_slot,
                                        DMD_TYP_MONITOR_INT64,
                                        MASTER_RTV_MON_REG_IDX, 0,
                                        DSA_DEF_TIMEOUT);
    if (err) {
        DSA_EXT_DIAG(err, master);
        return err;
    }
    err = dsa_assign_slot_to_register_s(grp, axis0_real_position_slot,
                                        DMD_TYP_MONITOR_INT64,
                                        SLAVES_RTV_MON_REG_IDX, 0,
                                        DSA_DEF_TIMEOUT);
    if (err) {
        DSA_EXT_DIAG(err, grp);
        return err;
    }
}

// Next function stops sharing real position ML1 of axis 0 to axes 1, 2, 3
// and UltimET
int unsharePositionAxis0(void)
```

```
{
    // Axes 1, 2, 3 free their own ML450 and UltimET frees its ML310
    err = dsa_unassign_slot_to_register_s(master, axis0_real_position_slot,
                                          DMD_TYP_MONITOR_INT64,
                                          MASTER_RTV_MON_REG_IDX, 0,
                                          DSA_DEF_TIMEOUT);
    if (err) {
        DSA_EXT_DIAG(err, master);
        return err;
    }

    err = dsa_unassign_slot_to_register_s(grp, axis0_real_position_slot
                                          DMD_TYP_MONITOR_INT64,
                                          SLAVES_RTV_MON_REG_IDX, 0,
                                          DSA_DEF_TIMEOUT);
    if (err) {
        DSA_EXT_DIAG(err, grp);
        return err;
    }

    // Axis 0 stops sharing its real position ML1 on TransnET
    err = dsa_unassign_register_to_slot_s(axis0, DMD_TYP_MONITOR_INT64,
                                          DMD_ML_POSITION_ACTUAL_VALUE_DS, 0,
                                          axis0_real_position_slot,
                                          DSA_DEF_TIMEOUT);
    if (err) {
        DSA_EXT_DIAG(err, axis0);
        return err;
    }

    err = dsa_free_64bit_rtv_slot(&axis0_real_position_slot);
    if (err) {
        DSA_EXT_DIAG(err, NULL);
        return err;
    }
}
```

14.4.3 SETRTV command

The **SETRTV** command allows the user to directly change a value in an RTV slot. This feature is helpful to quickly start a pending action on the AccurETs. The value is available in the AccurET controllers at the next TransnET cycle.

Command	***SETRTV = <P1>, <P2>**
Description	Sets an RTV slot P1 with a P2 value
Parameter	**Description**
<P1>	Slot: defines the slot in the TransnET cyclic data where the value is written.
<P2>	Value: defines the value (INT32) written in the slot.

Example:

```
// Several AccurET controllers (axes 1, 2 and 3) execute a sequence which waits
// for a change in their M450 monitoring register (pre-set using *ASR command)
// before starting their movement.

void func100(void)
{
    WPG.! = M450.0, 1;  // Wait for M450 greater than 1
    MVE.! = XL100.!;    // Move start
    ...
}

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

#define SLAVES_RTV_MON_REG_IDX  450
#define CMD_SETRTVVAL           532
```

```
// Consider:
// - A device group exists, made of axes 1, 2 and 3: grp
// - A master device exists, made of ULTIMET ADVANCED: master

DSA_RTV_SLOT* slotStartMove;      // Pointer on the RTV slot with the start
                                  // information
int startMoveSlotNbr = 0;

err = dsa_get_32bit_rtv0_slot(master, &slotStartMove);
if (err) {
    DSA_EXT_DIAG(err, master);
    return err;
}

err = dsa_get_32bit_rtv_slot_nr(slotStartMove, &startMoveSlotNbr);
if (err) {
    DSA_EXT_DIAG(err, master);
    return err;
}

// Axes 1, 2, 3 can access the start move field through their own M450
// register
err = dsa_assign_slot_to_register_s(grp, , slotStartMove,
                                    DMD_TYP_MONITOR_INT32,
                                    SLAVRS_RTV_MON_REG_IDX,
                                    0, DSA_DEF_TIMEOUT);
if (err) {
    DSA_EXT_DIAG(err, grp);
    return err;
}

// Write 0 on the slot
err = dsa_executeCommand_dd_s(master, CMD_SETRTVVAL, DMD_TYP_IMMEDIATE,
                              startMoveSlotNbr, DMD_TYP_IMMEDIATE, 0, FALSE,
                              TRUE, DSA_DEF_TIMEOUT);
if (err) {
    DSA_EXT_DIAG(err, master);
    return err;
}

// AccurET sequence waits for M450 to be at 1 before starting their movement.
err = dsa_execute_sequence_s(grp, 100, DSA_DEF_TIMEOUT);
if (err) {
    DSA_EXT_DIAG(err, grp);
    return err;
}

...

// Write 1 on the slot
err = dsa_executeCommand_dd_s(master, CMD_SETRTVVAL, DMD_TYP_IMMEDIATE,
                              startMoveSlotNbr, DMD_TYP_IMMEDIATE, 1,
                              FALSE, TRUE, DSA_DEF_TIMEOUT);
if (err) {
    DSA_EXT_DIAG(err, master);
    return err;
}
```

In the AccurET controllers' sequence, M450 becomes equal to one at the next AccurET cycle after the last `dsa_executeCommand_dd_s()` in the example above. Then all movements start.

Remark: Before restarting the AccurET sequence function, it is necessary to send the *SETRTV = slotStartMove, 0 command to avoid a premature start of the movements.+

14.5 PC (through PCIe) or CSM application data sharing functions

14.5.1 Introduction

The PCIe or CSM interfaces provide a direct access to TransnET. It is thus possible to get RTV values in a direct way. This kind of feature is mostly used in conjunction with PCs running a RTOS or a CSM running on the ULTIMET ADVANCED. In any case, it is possible to read / write RTV slots from a non-RTOS PCs as well.

14.5.2 Access to RTV slots from the application

A set of EDI functions (*) is dedicated to RTV configuration for read or write. These functions are high-level functions which combine several features. E.g `dsa_create_rtv_read` function which implements:

- Slot reservation.
- Register to slot assignment.

These high level functions can be combined with above mentioned functions, e.g. `dsa_assign_slot_to_register_s`.
(*) Refer to EDI documentation for more exhaustive description on all available RTV functions.

Example:

```
DSA_DRIVE *drv = NULL;            // AccurET object
int err = 0;
DSA_RTV_DATA *ML7_rtv = NULL;     // Realtime slot object referring to ML7 value
eint64 ML7;                       // Will contain read ML7 value

// Open Communication, ...

// Realtime slot object creation, referring to ML7
if (err = dsa_create_rtv_read(drv, DMD_TYP_MONITOR_INT64, 7, 0, &ML7_rtv)) {
    DSA_EXT_DIAG(err, drv);
    goto _error;
}

// Read RTV value in a loop, because the read can return DSA_RTVREADSYNCRO error
// which means that the read has access the RTV value exactly while the active
// RTV buffer has switched
printf("Read ML7 RTV value...\n");
do {
    if (err = dsa_read_rtv_int64(ML7_rtv, &ML7)) {
        if (err == DSA_ERTVREADSYNCRO)
            continue;
        DSA_EXT_DIAG(err, drv);
        goto _error;
    }
} while (err);
printf("->ML7 : %lld\n", ML7);

// Success
printf("success.\n");

// Error handler
_error:
if(dsa_is_valid_rtv(ML7_rtv))
    dsa_destroy_rtv(&ML7_rtv);
```

14.5.3 Cyclic RTV task

For PCs with a real-time OS or a CSM, it is possible to program one cyclic task with a user function called handler. Such handlers are used to:

- Read real-time values.
- Process these values.
- Write real-time values.

It is possible to have only one simultaneous cyclic handler.
Task cyclic rate may be configured between as a multiple of the TransnET period.

Typical use cases are:
- System regulation (secondary position loop, thermal regulation…).
- Real-time system monitoring.
- …

The functions below are used to:
- To configure and start a handler, to give it access to real-time data, to be read and / or written.
- To stop a cyclic handler wake-up task.
- To check if a cyclic handler is active or not.
- To enable – disable the watchdog associated with a handler.
- To get / reset execution time information of a running handler.

TransnET interrupt handler (requires an RTOS)	
int	dsa_start_rtv_handler (DSA_DSMAX *ultimet, int nr, int rate, DSA_RTV_HANDLER handler, int nb_read, DSA_RTV_DATA *read_rtv[], int nb_write, DSA_RTV_DATA *write_rtv[], void *user) Start a DSA_RTV_HANDLER real-time handler.
int	dsa_stop_rtv_handler (DSA_DSMAX *ultimet, int nr) Stop a DSA_RTV_HANDLER real-time handler.
int	dsa_get_rtv_handler_activity(DSA_MASTER *master, int nr, ebool *active); Check if DSA_RTV_HANDLER real-time handler attached to cyclic interrupt "nr" is active
int	dsa_set_rtv_handler_watchdog(DSA_MASTER *master, int rtv_id, ebool enable); Enable or Disable the watchdog for the cyclic RTV handler
int	dsa_get_rtv_handler_times(DSA_MASTER *master, int nr, double *max_time, double *min_time, double *mean_time); Get the execution time of the rtv handler function
int	dsa_reset_rtv_handler_times(DSA_MASTER *master, int rtv_id); Reset the execution time measurements of the rtv handler function

Example:

```
// This example shows how to write a cyclic handler, how to enable its related
// watchdog, and how to start and stop it.

// This structure contains pointers on axes conversion factor, axes previous
// position, and the number of axes
typedef struct {
    int nbAxes;
    double* positionIsoConversionFactor[];
    double* prevPosition[]
} PosStruct;

double positionIsoConversionFactors[NB_AXES];
double prevPositions[NB_AXES];
PosStruct posData;
posData.nbAxes = NB_AXES;
posData.positionIdoConversionFactor = &positionIsoConversionFactors[0];
posData.prevPosition = &prevPositions[0];
DSA_MASTER* master = NULL;
DSA_RTV_DATA* readPositionData[NB_AXES];
DSA_RTV_DATA* writeVelocityData;
int transnETCyclesRate = 1;
int rtvInterruptNumber = 0;


// RTV handler
// This handler will be called by the EDI kernel at the rate specified by
// dsa_start_rtv_handler. In our case, this handler will be called every 100 µs.
// nb_read axes positions are read and converted in iso.
// The curvilinear velocity is calculated.
static void DSA_CALLBACK rtv_handler(DSA_MASTER *ultimet, int nr, int nb_read,
                                     DSA_RTV_DATA *read_table[], int nb_write,
                                     DSA_RTV_DATA *write_table[], void *user)
{
    int err;
    double axisRealPos[NB_AXES];
    double f64Val = 0;
```

```
        eint64 i64Val = 0;
        int i;

        // Map the user callback variable as a PosStruct
        PosStruct* localPosDataPtr = (PosStruct *)user;

        // Check array sizes
        if (localPosConv->nbAxes < nb_read)
        {
            // Error handling, avoid "printf"
            goto _error;
        }

        // Get axes position, convert it into iso, and calculate the curvilinear
        // position.
        for (i=0;i< nb_read;i++)
        {
            if (err == dsa_read_rtv_int64(read_table [i], &i64Val))
            {
                // Error handling, avoid "printf"
                goto _error;
            }

            axisRealPos[i] = (double)i64Val *
                             (*localPosDataPtr->positionIsoConversionFactor[i]);
            f64Val += pow(axisRealPos[i] - localPosDataPtr-> prevPosition[i], 2.0);
            localPosDataPtr->prevPosition[i] = axisRealPos[i];
        }

        f64Val = sqrt(f64Val);

        // Return this position in a 64bit slot
        if (err = dsa_write_rtv_float64(write_table[0], f64Val))
        {
            // Error handling, avoid "printf"
            goto _error;
        }

    _error :
        return;
    }

    // In the main program, consider:
    // - the communication is already opened,
    // - the conversion factors have been calculated
    // - the initial positions are stored in PosData
    // - the readPositionData and writeVelocityData already configured
    //
    // Enable the watchdog for this cyclic handler to be started (must be done prior
    // to start the handler)
    if (err = dsa_set_rtv_handler_watchdog(master, 0, TRUE))
    {
        // Error handling
        DSA_EXT_DIAG(err, master);
    }
    // Start the handler at a rate of "rtvInterruptNumber" TransnET cycle
    if (err = dsa_start_rtv_handler(master, rtvInterruptNumber, transnETCyclesRate,
                                    rtv_handler,NB_AXES,
                                    DSA_RTV_DATA readPositionData,
                                    1, writeVelocityData, & posData))
    {
        // Error handling
        DSA_EXT_DIAG(err, master);
    }

    // Wait for a process information to stop the handler

    // Stop the handler
    if (err = dsa_stop_rtv_handler(master, rtvInterruptNumber))
    {
        // error handling
        DSA_EXT_DIAG(err, master);
    }
```

14.6 Real-time channels timing

Here are the different delays and update rates on real time channels:

Configuration	Delay	Update rate
AccurET <=> AccurET	150 µs (worst case)	100 µs
AccurET <=> UltimET	200 µs (worst case)	100 µs
AccurET (via UltimET PCIe) <=> PC	PC dependent	100 µs
UltimET (PCIe) <=> PC	PC dependent	100 µs
AccurET (via UltimET) <=> CSM	200 µs (worst case)	100 µs
UltimET <=> CSM	100 µs (worst case)	100 µs

15 Interpolation

15.1 Introduction

To be able to use the commands described in this chapter it is important to have an overview of how the interpolation command processing works. 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 monitor 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 ADVANCED. The commands related to the movement are pre-processed at the command manager level and then stored in a buffer (refer to §15.1.2 for more information about the buffer) and then processed by the trajectory generator.

Commands operate on what is called an interpolation group. An interpolation group is a set of 4 logical axes: X, Y, Z and Θ. Each logical axis can be made up of zero or several physical axes. One typical example is a gantry system where the logical X axis is made up of the two axes making up the gantry, the beam would be the Y axis, and there could be a rotary motor on the beam. In this system the Z logical axis is “empty”.

Remark: All 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 ADVANCED commands list in §29.

15.1.1 Management of the interpolation commands

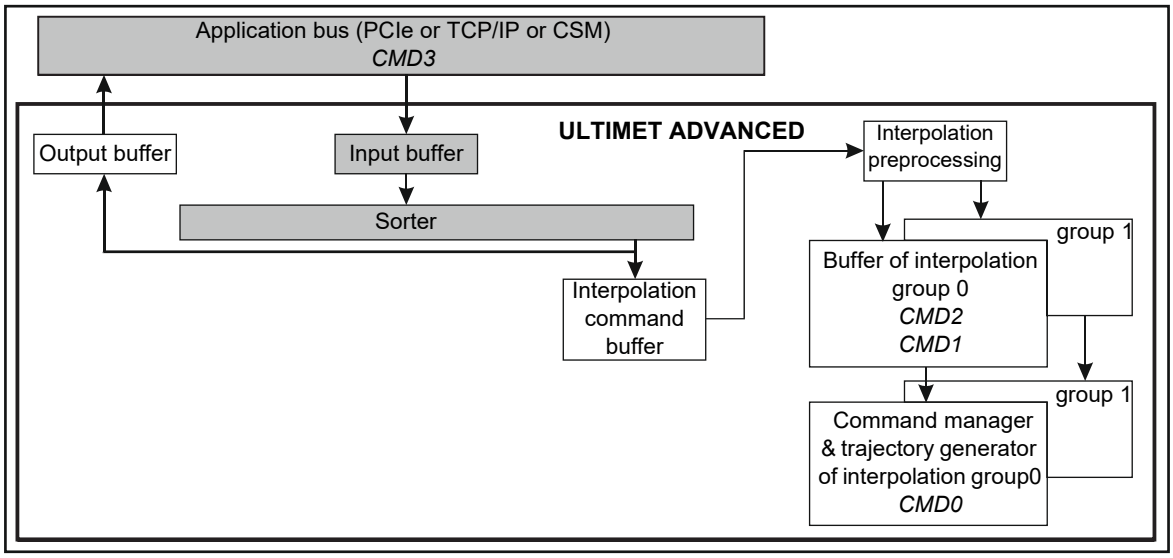
The processing of interpolation commands (*ILINE, *ICWR, *IMARK,...) is slightly different from the others. When a command arrives from the PCIe or TCP/IP ports or from a CSM, it is preprocessed and put in the corresponding interpolation buffer (1 buffer per interpolation group) and is considered as finished (at the PCIe, TCP/IP or CSM level). It means that the acknowledgment is immediately returned to the PCIe, TCP/IP port or CSM.

The trajectory calculator then takes the commands from the buffers, one command after the other in the order of arrival to execute them. It is at this point in time that the motors are going to move according to the movement defined by the command.

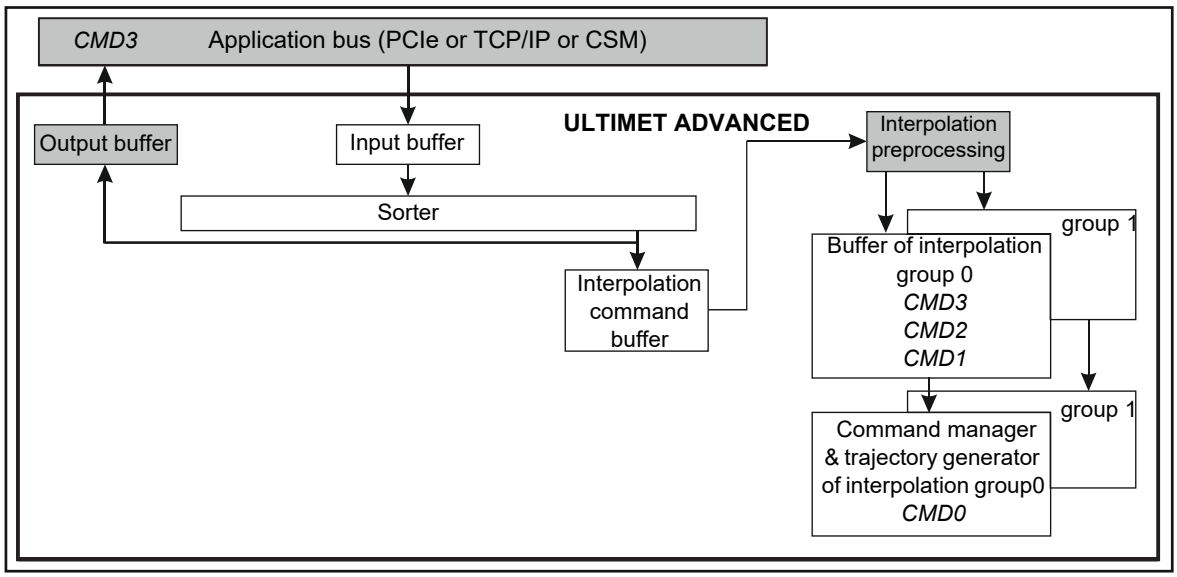
The time between the acknowledgment of the command and its real execution is equal to the sum of all the movements’ times defined by the commands already present in the buffer.

Example:

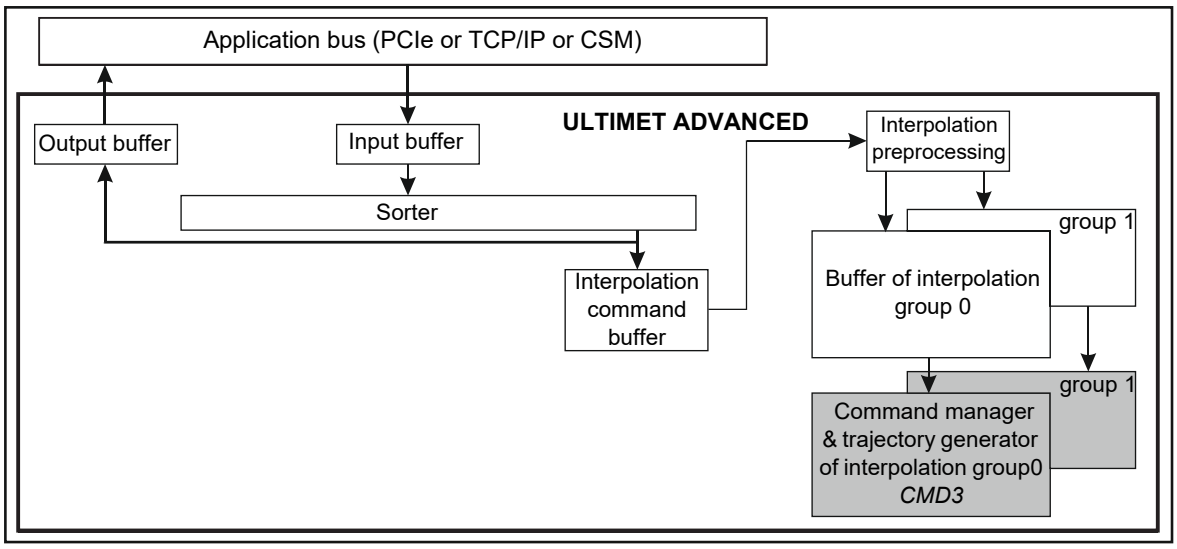
Here is the progression of an interpolation command. First, CMD3 command, intended for the interpolation group 0 and coming from the PCIe, TCP/IP port or CSM, is preprocessed and goes to the interpolation buffer of the group 0, which already contains the CMD1 and CMD2 commands while the trajectory calculator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



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



When the trajectory calculator will have finished executing 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 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 register ***M513** gives the equivalent in number of buffer slots. The monitoring register ***M512** gives the actual number of free buffer 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.

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

M	Comment
***M512**	Actual number of remaining free buffer slots.
***M513**	Number of free buffer 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 ADVANCED commands list in §29..

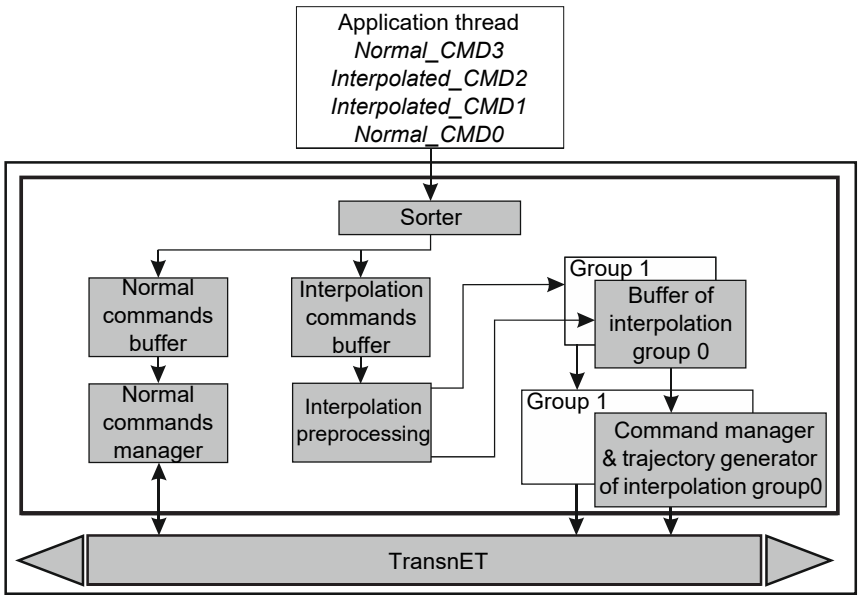
The monitoring register ***M535** allows the user to know the number of commands present in the interpolation buffer.

M	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 (§15.4.4.2).

15.1.2 Management of a mix of normal and interpolated commands

Consider a series of synchronous commands coming from one thread of an application. This application thread contains normal commands (e.g. power_on, set_register, …) and interpolated commands (ILINE, IPVT,...), that are placed in a certain order.



The application thread dispatches commands on specific buffers, based on their type: normal or interpolated. This is done sequentially (synchronous commands). Normal commands are stored in the normal commands buffer. They are then processed by the normal commands manager. Interpolated commands are stored in the interpolation commands 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 commands manager: if the command concerns the UltimET (*X1=10,...), it is directly executed. If it concerns the axis (DOUT.1=2,...) it is sent through the TransnET to the specific controller(s).

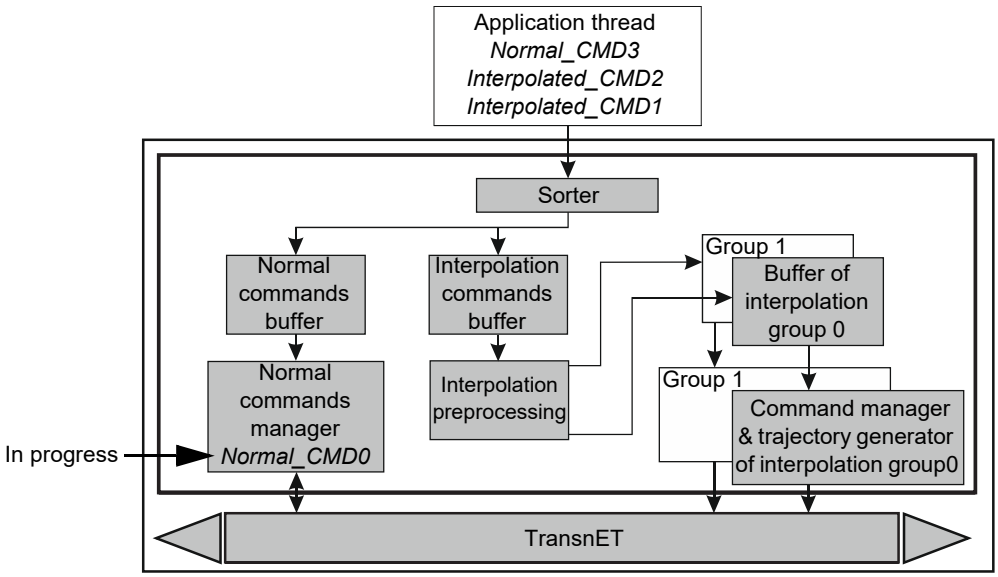
- Interpolation manager: the trajectory command is handled by the UltimET trajectory generator and target positions are sent to the controller(s) that are concerned by the interpolation group (as defined in *ISET command).

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

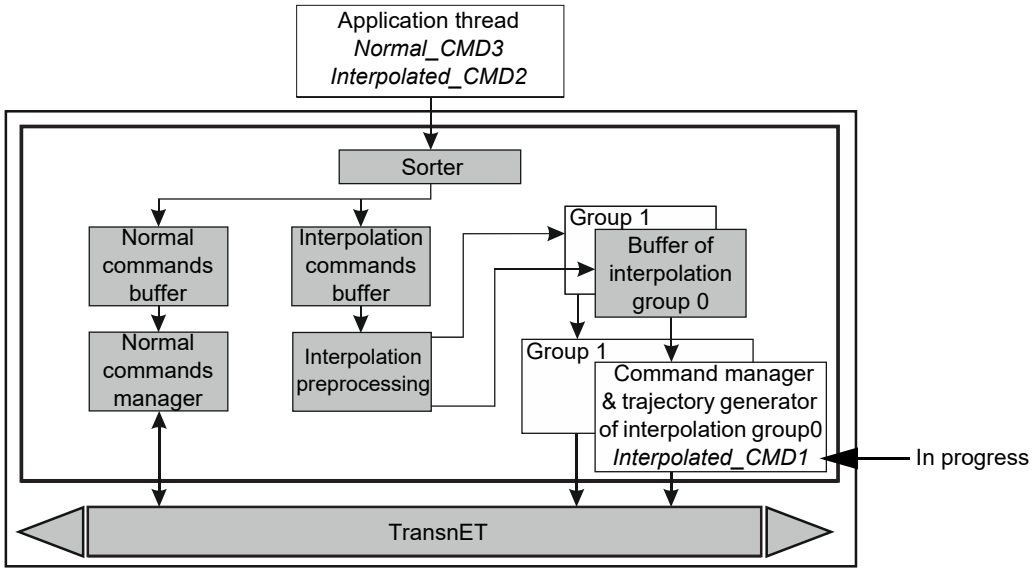
15.1.2.1 Command processing order

The commands are processed in the following order:

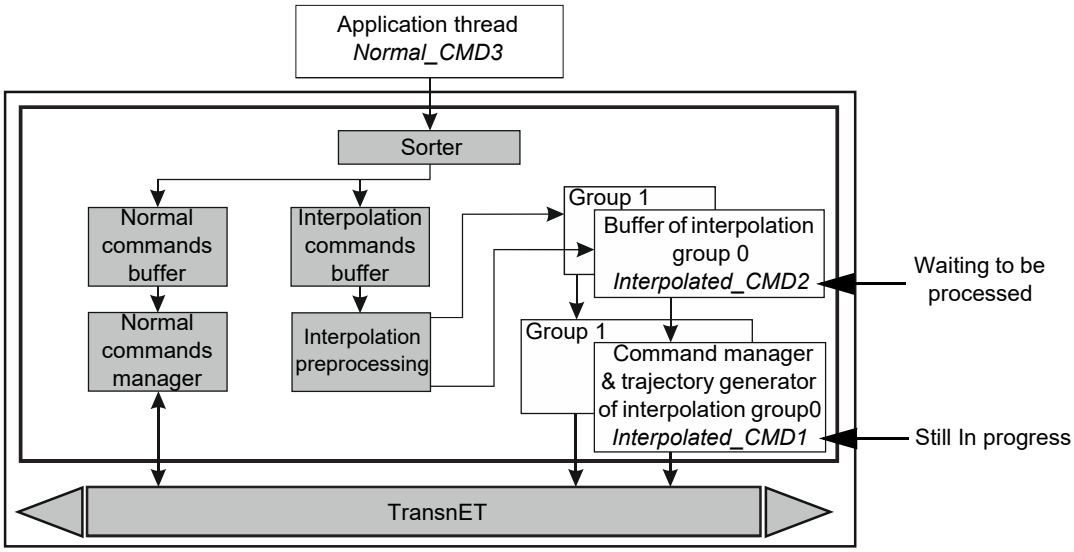
1. Normal CMD 0 is first stored in the normal commands buffer. As soon as the normal commands manager sees it, CMD 0 is processed and then acknowledged at the application level.



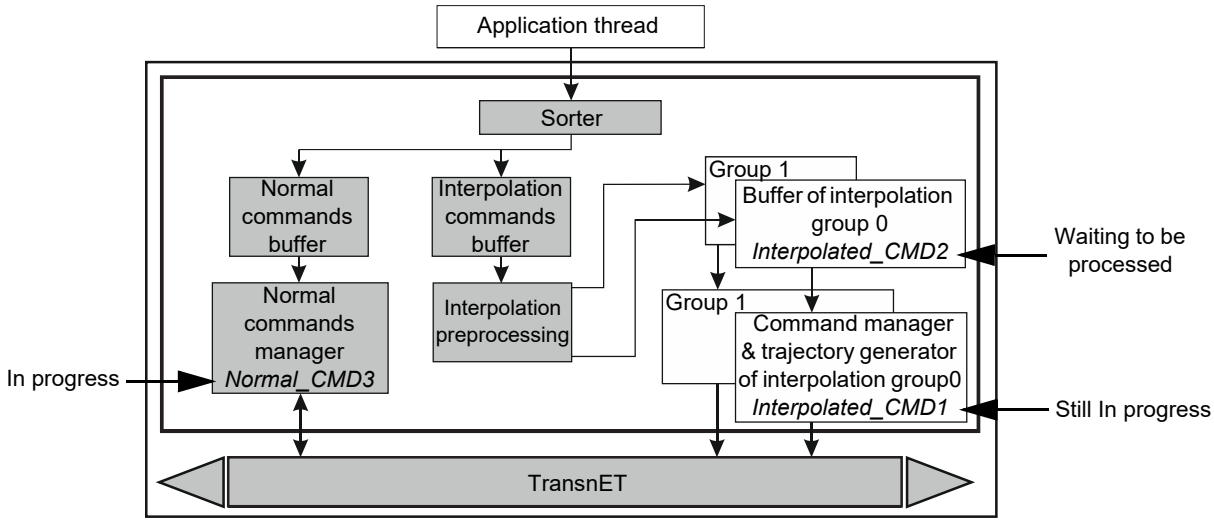
2. Interpolated CMD 1 is pre-processed and stored in the interpolation group 0 buffer. As soon as the interpolation commands manager sees it, CMD 1 starts to be processed.



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 sees it, CMD 3 is processed and then acknowledged at the application level.



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

15.1.3 Units of the interpolation commands

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

- For position: meters for a linear motor and turns for a rotary motor.
- For speed: meters/second for a linear motor and turns/second for a rotary motor.
- For acceleration: meters/second2 for a linear motor and in turns/second2 for a rotary motor.
- For time: seconds.

The power of ten can be defined by the user and in K parameters (***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 ADVANCED. These units have a direct influence also on the internal resolution of the ULTIMET ADVANCED 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.

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

The units are affected as follows:

- Unit for position in interpolated command = *K522 x 10^{*K523} [meters or turns] = ufpi (user friendly position increment).
- Unit for speed in interpolated command = 10^{*K524} [meters / second or turns / second] = ufsi (user friendly speed increment).
- Unit for acceleration in interpolated command = 10^{*K525} [meters / second^2 or turns / second^2] = ufai (user friendly acceleration increment).
- Unit for time in interpolated command = 10^{*K526} [seconds] = ufti (user friendly time increment).

Example:

To work with a linear system with the following units for the interpolation group 0:
• position in tenth of micrometers
• speed in micrometers/second
• acceleration in micrometers/second2
• time in milliseconds

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
```

15.1.4 Distance and time limits for interpolation commands

Most of the interpolation commands need parameters representing a distance. All these distances have to be within 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 register ***M659**, results from the definition of the position unit in the ULTIMET ADVANCED (*K522 and *K523 parameters), the controllers' position resolutions and the maximum value of a 32 bits signed integer. The controller resolution used in the formula below 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 in the ULTIMET ADVANCED.

The maximum length of a segment is limited by this value:
ctrl_max_length = highest controller resolution [m] x (2^{32}-1).

The maximum position value in the ULTIMET ADVANCED depends on the units definition:
UA_max_length = *K522 x 10^{*K523} x (2^{31}-1).

All distance and position values passed as parameters of all interpolation commands are limited by the minimum of the above two values:
Max_distance [m] = minimum (ctrl_max_length, UA_max_length,).

The minimum distance is:
Min_distance = - max_distance

The maximum time value, given by the monitoring register ***M658**, results from the definition of the time unit in the ULTIMET ADVANCED (*K526 parameter).

Max_time [s] (=*M658) = 10^{*K526} x (2^{31}-1)).

M	Comment
***M658**	Maximum time value for interpolation commands (unit: ufti); depth 0 for group 0 and depth 1 for group 1.
***M659**	Maximum distance for interpolation commands (unit: ufpi); depth 0 for group 0 and depth 1 for group 1.

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:
Max_distance = minimum (controller resolution x (2^{31}-1); *K522 x 10^{*K523} x (2^{31}-1))
= minimum (1 x 10^{-6} / 1024 x (2^{31}-1); 1 x 10^{-6} x (2^{31}-1))
= minimum (2.097; 2147)
= 2.097 m

Min_distance = - max_distance = -2.097 m

Example 2:

For a single axis with a low resolution of 256 increments per millimeter and a user position unit of one micrometer (*K522:0 = 1, *K523:0 = -6), the position limits are:
Max_distance = minimum (controller resolution x (2^{31}-1); *K522 x 10^{*K523} x (2^{31}-1))
= minimum (1 x 10^{-3} / 256 x (2^{31}-1); 1 x 10^{-6} x (2^{31}-1))
= minimum (8388; 2147)
= 2147 m

Min_distance = - max_distance = -2147 m

15.2 Setting of the interpolation

This section describes the commands which enable the definition, activation and deactivation of an interpolation group:

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

The last subsection of this chapter describes EDI related functions.

15.2.1 ISET command

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. This defines the user reference frame for a group. The nodes associated to the interpolated axes belong to the interpolation group determined by the ipol_group parameter.

The three axes (axis_x, axis_y, axis_z) normally define a three-dimensional orthonormal and Cartesian reference frame. The 'axis_theta' 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. Out of plane circular movements are feasible by using the matrix functions (refer to §15.7 for more information).

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

Command	***ISET = <P1>, <P2>, <P3>, <P4>, <P5>**		
Description	Defines the nodes associated with the different axes of a specified interpolation group.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group.
<P2>	Axis_x	From 0 to (2^{64}-1)	Mask of the nodes associated to the X interpolated axis (integer 64-bit value).
<P3>	Axis_y	From 0 to (2^{64}-1)	Mask of the nodes associated to the Y interpolated axis (integer 64-bit value).
<P4>	Axis_z	From 0 to (2^{64}-1)	Mask of the nodes associated to the Z interpolated axis (integer 64-bit value).
<P5>	Axis_θ	From 0 to (2^{64}-1)	Mask of the nodes associated to θ interpolated axis (integer 64-bit value).

Example:

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

The node 0 makes up axis_x, the nodes 3 and 8 make up axis_y and the node 12 makes up axis_z. The command line is as follows:

```
*ISET = 0, 1L, 264L, 4096L, 0L; // The interpolated group number is 0
                                // The mask for the axis_x is 0x1L = 20
                                // The mask for the axis_y is 0x108L = 23 + 28
                                // The mask for the axis_z is 0x1000L = 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 nodes 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 encoder parameters.**
- **The encoder interpolation parameters must also be identical (refer to the AccurET 'Operation & Software Manual' for more information about these parameters).**

A copy of these parameters is automatically stored in the M and ML monitoring registers (on the ULTIMET ADVANCED) as well as in the nodes of the masks of the associated axes. A monitoring register also defines if the interpolated axis is activated or not.

The configuration monitoring registers are:

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

*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 controller data are automatically read when the ISET command is executed.

15.2.2 IBEGIN command

The **IBEGIN** command allows the user to switch into interpolated mode, for the group specified as parameter.

Command	***IBEGIN = <P1>**		
Description	Switches the selected interpolation group into interpolated mode.		
Parameter	**Description**	**Value**	**Description**
<P1>	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 ADVANCED.

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

M	Value	Comment
***M530**	0 1	The interpolated mode is not active The interpolated mode is active

Note that the jerk time (defined in the controllers by parameter K213) and the cam value (defined in the controllers by parameter KF205) 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 ADVANCED is in error. First of all, find the reason of the error, correct and reset it (with the *RST command).

15.2.2.1 Origin point of the interpolation mode

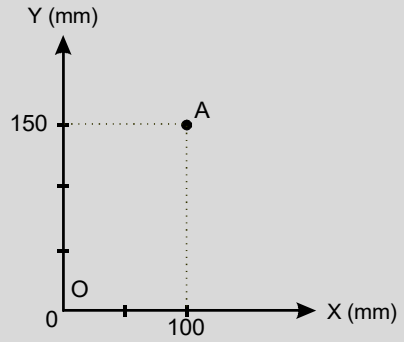
In a multi-axis system, each axis has a specific zero position. A standard reference frame (also called 'Machine reference frame') is defined by the axes orientation, and the origin of that reference frame corresponds to the point where each axis theoretical position is equal to 0.
In the interpolated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

- **Relative mode**

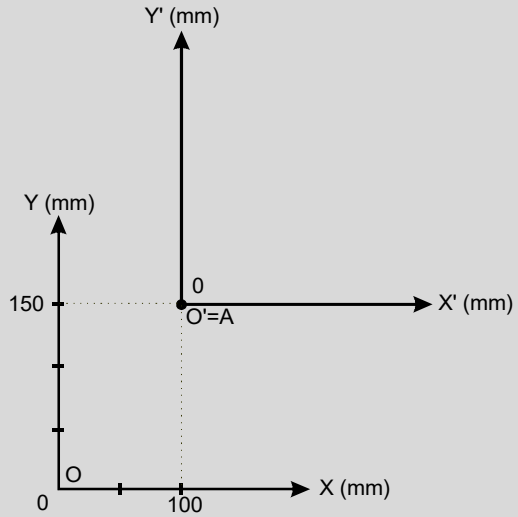
This mode is the default mode. The origin point of the interpolation mode reference frame (also called 'User reference frame') is defined at the time of execution of the IBEGIN command: before this command, each axis has a specific theoretical position. All these theoretical positions define a point in the machine reference frame. At the time of execution of the IBEGIN command, that point becomes the origin point of the interpolation mode reference frame. As the origin points of these two reference frames may be different, there can be 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.

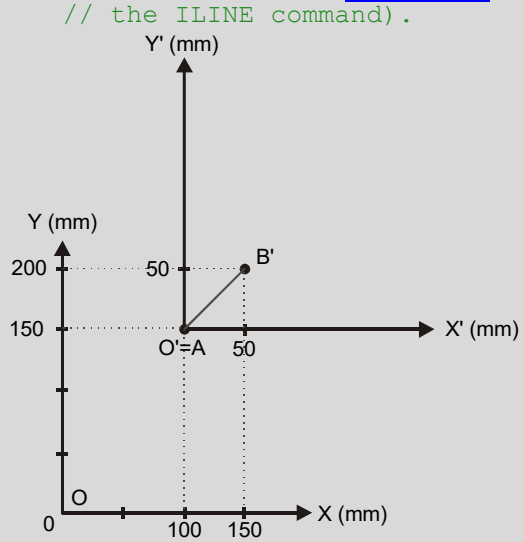


When the IBEGIN command is executed at this point, it defines the origin point of the interpolated mode reference frame O'X'Y'. Point A in the machine reference frame is then equal to point O' (X'=0 mm, Y'=0 mm) in the interpolated reference frame.



A linear interpolated movement is specified and executed in the interpolation mode reference frame (user position unit in millimeter `*K522:0 = 1, *K523:0 = -3`):

```
*ILINE = 0, 50, 50, 0, 0; // Linear displacement to the point B' (X'=50 mm, Y'=50
                          // mm). Refer to §15.4.1.1 for more information about
                          // the ILINE command).
```



In the machine reference frame, the coordinates of 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**

If there are more than one node per interpolated axis, it is the origin point of the first one that is taken into account.

The **IABSMODE** (**I**nterpolation **ABS**olute **MODE**) command allows the user to activate or clear the absolute reference coordinates mode of the selected interpolation group. It must be called after the IBEGIN command.

Command	***IABSMODE = <P1>, <P2>**		
Description	Activates (1) or clears (0) the absolute reference coordinates mode of the selected interpolation group.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group
<P2>	Reference_mode	0 1	Clears the absolute mode Activates the absolute mode

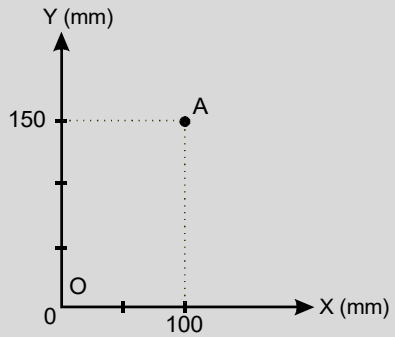
Remark: When the reference_mode is equal to 0, the active mode thereafter is the relative mode.

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

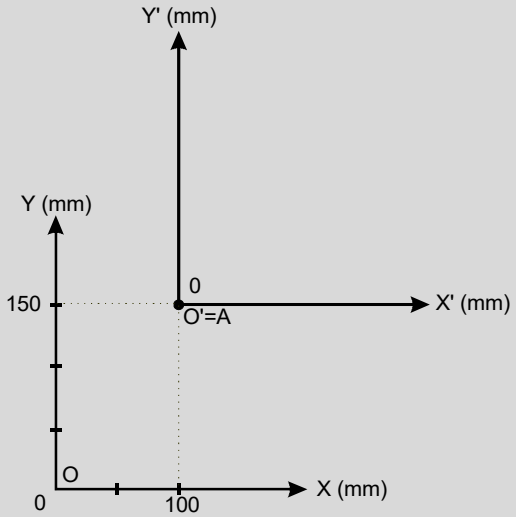
M	Value	Comment
***M590**	0 1	Absolute mode deactivated. It corresponds then to the relative mode Absolute mode activated

Example:

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

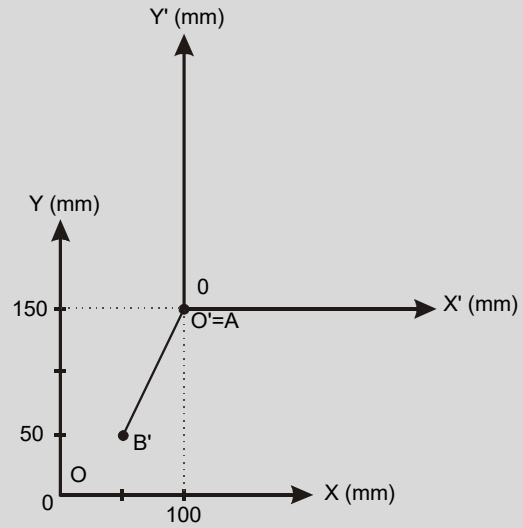


When the IBEGIN command is executed at this point, it defines the origin point of the interpolated mode reference frame O'X'Y'. Point A in the machine reference frame is then equal to point O' (X'=0 mm, Y'=0 mm) in the interpolated reference frame.




```
*IABSMODE = 0, 1                // The absolute mode is activated
```

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



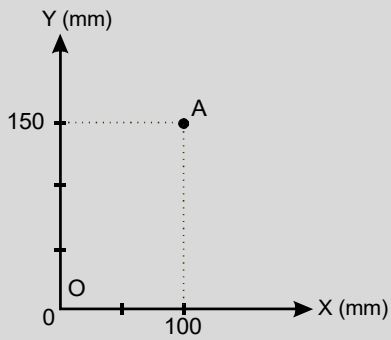
- **Absolute mode with the interpolation origin point defined by the user**

The **IABSCOORDS** (**I**nterpolation **ABS**olute **COORD**inate**S**) command allows the user to define the new origin point by specifying the coordinates of the current position of the axes in the future reference frame. It must be called after the IBEGIN command.

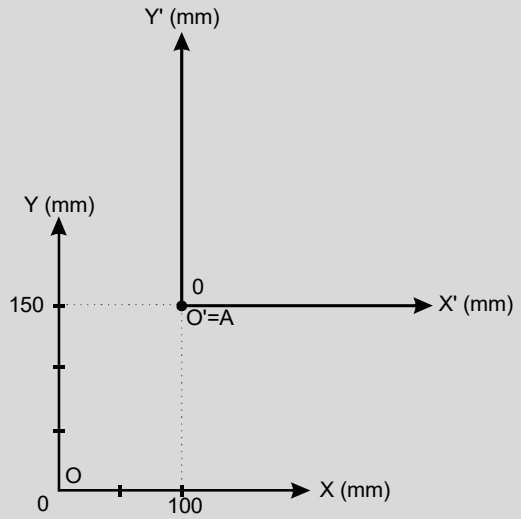
Command	***IABSCOORDS = <P1>, <P2>, <P3>, <P4>, <P5>**		
Description	Defines the coordinates of the current position of the axes of the selected interpolation group as the new origin of the coordinates frame		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group
<P2>	Current_coord_x	From -distance_max to +distance_max (**)	x axis absolute coordinates
<P3>	Current_coord_y	From -distance_max to +distance_max (**)	y axis absolute coordinates
<P4>	Current_coord_z	From -distance_max to +distance_max (**)	z axis absolute coordinates
<P5>	Current_coord_θ	From -distance_max to +distance_max (**)	θ axis absolute coordinates

Example:

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

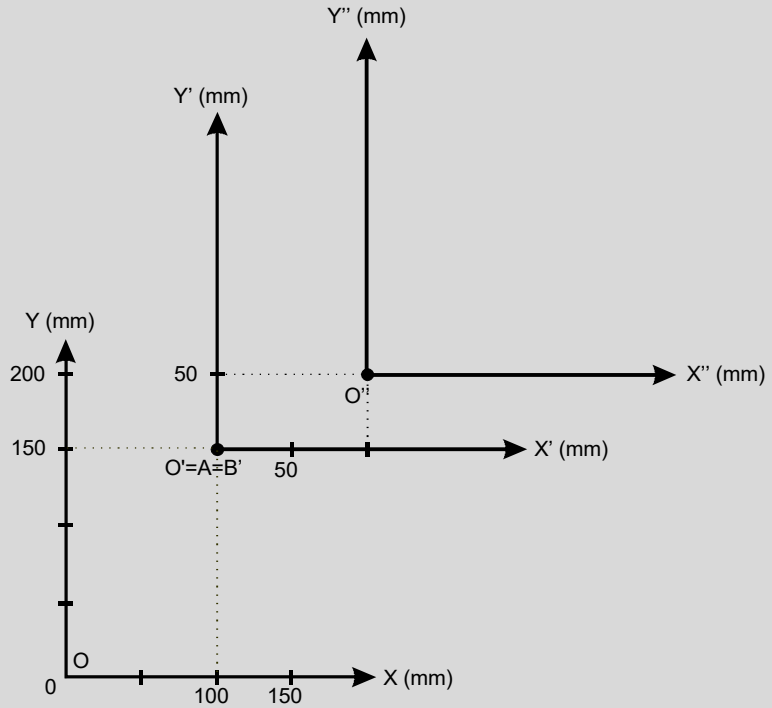


When the IBEGIN command is executed at this point, it defines the origin point of the interpolated mode reference frame O'X'Y'. Point A in the machine reference frame is then equal to the point O' (X'=0 mm, Y'=0 mm) in the interpolated reference frame.

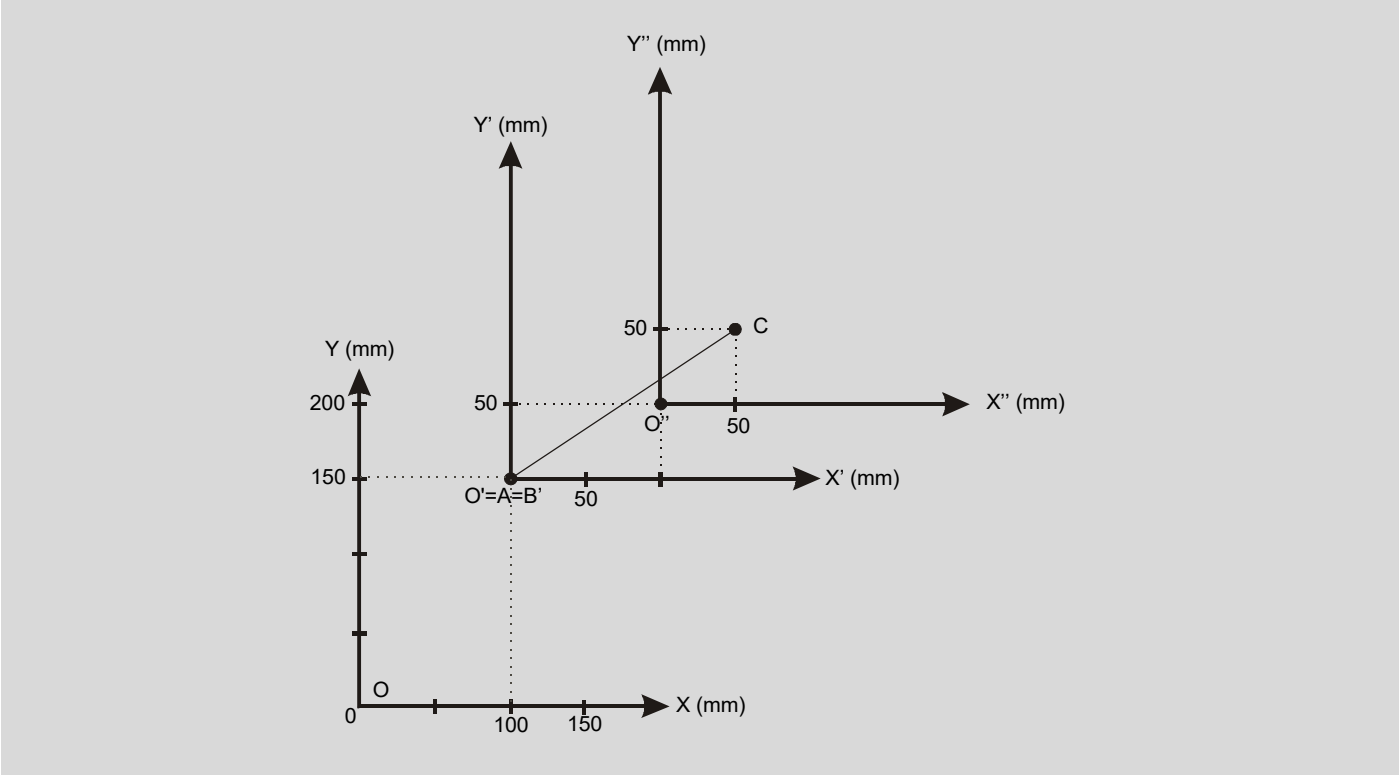



```
*IABSCOORDS = 0, -100, -50, 0, 0; // The current point coordinates (and then
                                  // the interpolation mode reference frame) is
                                  // set at point B' (X'=-100 mm, Y'=-50 mm)
```

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 point C (X''=50
                               // mm, Y''=50 mm)(Refer to §15.4.1.1 for more
                               // information about the ILINE command).
```



15.2.3 IEND command

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

Command	***IEND = <P1>**		
Description	Switches the selected interpolation group out of interpolated mode.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	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 ADVANCED.

As *IEND command is immediately executed, the current movement must be finished before sending this command for it to terminate smoothly.

15.2.4 Two groups of interpolation

When working with two interpolation groups, a 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 every 200 µs.

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

Caution: The *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 value defined for *K600≠0, 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.

15.2.5 EDI related functions

Referring to the example shown for the IABSCOORDS command (§15.2.2.1), here is the EDI version of this example.

Example:

```
// Create the interpolation group object and set it up.
if (err = dsa_create_ipol_group(&igrp, 2)) {
    DSA_EXT_DIAG(err, igrp);
    goto _error;
}
if (err = dsa_set_group_item(igrp, 0, x)) {
    DSA_EXT_DIAG(err, igrp);
    goto _error;
}
if (err = dsa_set_group_item(igrp, 1, y)) {
    DSA_EXT_DIAG(err, igrp);
    goto _error;
}

// Make a movement to a specified point.
if (err = dsa_set_target_position_s(x, 0, 0.01, DSA_DEF_TIMEOUT)) {
    DSA_EXT_DIAG(err, x);
    goto _error;
}
if (err = dsa_set_target_position_s(y, 0, 0.015, DSA_DEF_TIMEOUT)) {
    DSA_EXT_DIAG(err, y);
    goto _error;
}
// Wait for the end of the movement on both axes. Here, the interpolation group
// is used like a normal group. The ULTIMET associated with this group is not
// taken into account.
if (err = dsa_wait_movement_s(igrp, 60000)) {
    DSA_EXT_DIAG(err, igrp);
    goto _error;
}

// Interpolated mode will be activated.
// The X-Y system is in the following absolute drive position (in meters):
// x = 0.01, y = 0.015
// When entering into interpolated mode, the actual position (x = 0.01, y =
// 0.015) becomes the origin (0, 0) for interpolation. For the interpolation
// functions (dsa_ipol_xxx()), all coordinates are given in absolute from this
// origin point.
if (err = dsa_ipol_begin_s(igrp, DSA_DEF_TIMEOUT)) {
    DSA_EXT_DIAG(err, igrp);
    goto _error;
}

// Define the maximum tangential speed and acceleration that must be performed
// during the interpolation.
// The tangential speed is set to 0.05 m/s.
// The tangential acceleration and deceleration are set to 0.1 m/(s*s).
if (err = dsa_ipol_tan_velocity_s(igrp, 0.05, DSA_DEF_TIMEOUT)) {
    DSA_EXT_DIAG(err, igrp);
    goto _error;
}
if (err = dsa_ipol_tan_acceleration_s(igrp, 0.1, DSA_DEF_TIMEOUT)) {
    DSA_EXT_DIAG(err, igrp);
    goto _error;
}
if (err = dsa_ipol_tan_deceleration_s(igrp, 0.1, DSA_DEF_TIMEOUT)) {
    DSA_EXT_DIAG(err, igrp);
    goto _error;
}

// Change the current point coordinates as (X=-0.01m and Y=-0.005m) instead of
// X=0, Y=0) and set the interpolation mode reference frame origin at drives
// coordinates (X=0.02m, Y=0.02m).
absoluteCorrdinates.size= sizeof(DSA_VECTOR);
absoluteCorrdinates.val.dim.x       = -0.01;
absoluteCorrdinates.val.dim.y       = -0.005;
absoluteCorrdinates.val.dim.z       = 0.0;
absoluteCorrdinates.val.dim.theta   = 0.0;
if (err = dsa_ipol_abs_coords_s(igrp, &absoluteCorrdinates, DSA_DEF_TIMEOUT))
{
    DSA_EXT_DIAG(err, igrp);
    goto _error;
```

```
        }

        // Move to interpolation mode reference frame coordinates (X=0.005m, Y=0.005m)
        if (err = dsa_ipol_line_2d_s(igrp, 0.005, 0.005, DSA_DEF_TIMEOUT)) {
            DSA_EXT_DIAG(err, igrp);
            goto _error;
        }

        // Wait for the end of the interpolated movement.
        if (err = dsa_ipol_wait_movement_s(igrp, 60000)) {
            DSA_EXT_DIAG(err, igrp);
            goto _error;
        }

        // Validation: Final coordinates must be:
        // Ipol coordinates: (X=0.005m, Y=0.005m)
        // drive coordinates: (X=0.025m, Y=0.025m)
```

15.3 User position vs. real position

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

M	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)

15.4 Interpolation movements

Three different interpolation movements can be realized with an ULTIMET ADVANCED:

- '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).

Trajectories are made up of segments, a segment being the displacement created by one single interpolation movement command.

15.4.1 'G-code based' mode

The G-code defines the coding of basic 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 the commands of the movement generation are defined. 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 ADVANCED 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 ADVANCED implements 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 desired type of segment and 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 ICCWR or ICCW commands are actually ETEL commands for G01, G02 and G03. ISET and IBEGIN commands must be executed before executing the commands described in the following chapter.

15.4.1.1 Linear interpolated movement command

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

Command	***ILINE = <P1>, <P2>, <P3>, <P4>, <P5>**		
Description	Adds a linear move to the trajectory for the selected interpolation group.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group
<P2>	Coord_x	From -distance_max to +distance_max (*)	x coordinates of the target point
<P3>	Coord_y	From -distance_max to +distance_max (*)	y coordinates of the target point
<P4>	Coord_z	From -distance_max to +distance_max (*)	z coordinates of the target point
<P5>	Coord_θ	From -distance_max to +distance_max (*)	θ coordinates of the target point

(*): Refer to §15.1.3 for more information.

From an EDI application:

```
int dsa_ipol_line_s(DSA_IPOL_GROUP *igrp, DSA_VECTOR *dest, long timeout);
```

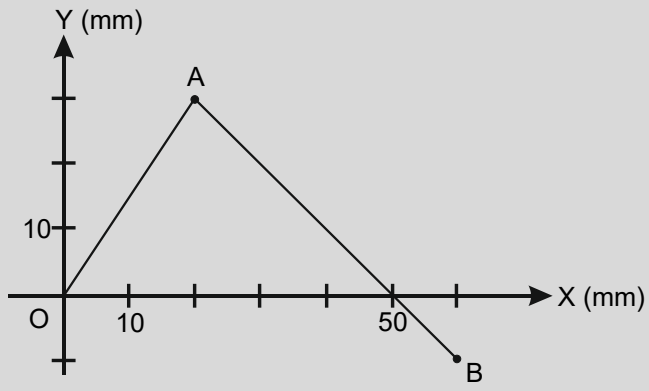
A movement profile is applied on this movement Refer to §15.4.1.3 for more information.

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=0 mm, Y=0 mm) to point A (X=20 mm, Y=30 mm) and the second goes from point A to point B (X=60 mm, Y=-10 mm). 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;
```



15.4.1.2 *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).

Command	***ICW (or *ICCW) = <P1>, <P2>, <P3>, <P4>, <P5>**		
Description	Adds a ClockWise (Counter-ClockWise) circular move, with center definition to the trajectory for the selected interpolation group.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group
<P2>	Coord_x	From -distance_max to +distance_max (*)	x coordinates of the target point
<P3>	Coord_y	From -distance_max to +distance_max (*)	y coordinates of the target point
<P4>	Center_x	From -distance_max to +distance_max (*)	x coordinates of the circle center
<P5>	Center_y	From -distance_max to +distance_max (*)	y coordinates of the circle center

(*): Refer to §15.1.3 for more information.

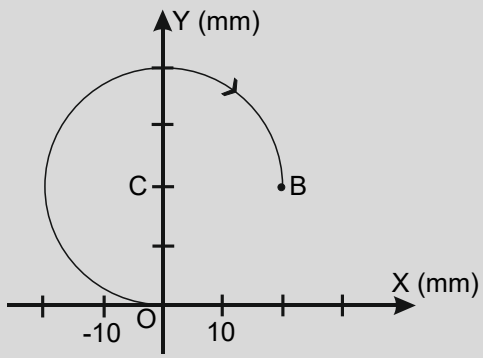
From an EDI application:

```
int dsa_ipol_circle_cw_c2d_s(DSA_IPOL_GROUP *igrp, double x, double y,
                             double cx, double cy, long timeout);

int dsa_ipol_circle_ccw_c2d_s(DSA_IPOL_GROUP *igrp, double x, double y,
                              double cx, double cy, long timeout);
```

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 the center of the circle” and “end point to the 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 joins the badly defined starting point to the corrected one.

Example:

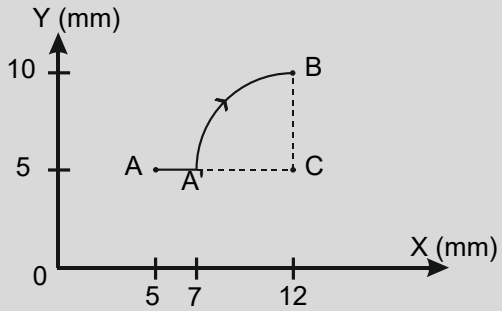
In an XY stage, the current point is A (X=5mm, Y=5mm). The interpolated mode is active and the user position unit is the micron. The following command is sent:

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

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 ADVANCED automatically introduces a new command line before the above-mentioned command. The executed sequence is:

```
*ILINE = 0, 7000, 5000, 0, 0;        // Linear movement to point A'(X=7mm, Y=5mm)
*ICW = 0, 12000, 10000, 12000, 5000; // Clockwise circle to reach point B (X=12mm,
                                     // Y=10mm) with a circle given by point
                                     // C (X=12mm, Y=5mm)
```

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

Command	*ICWR (or *ICCWR) = <P1>, <P2>, <P3>, <P4>		
Description	Adds a ClockWise (Counter-ClockWise) circular move, with radius definition to the trajectory for the selected interpolation group		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group
<P2>	Coord_x	From -distance_max to +distance_max (*)	x coordinates of the target point
<P3>	Coord_y	From -distance_max to +distance_max (*)	y coordinates of the target point
<P4>	Radius	From -distance_max to +distance_max (*)	Length of the radius of the circle

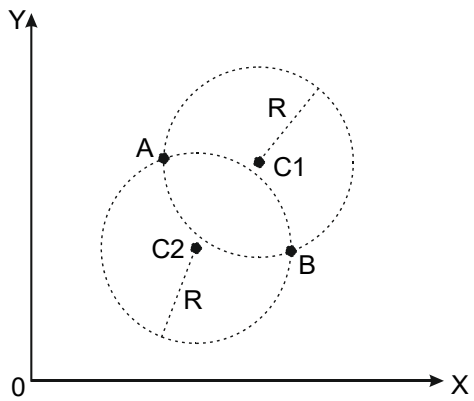
(*): Refer to §15.1.3 for more information.

From an EDI application:

```
int dsa_ipol_circle_cw_r2d_s(DSA_IPOL_GROUP *igrp, double x, double y,
                             double r, long timeout);

int dsa_ipol_circle_ccw_r2d_s(DSA_IPOL_GROUP *igrp, double x, double y,
                              double r, long timeout);
```

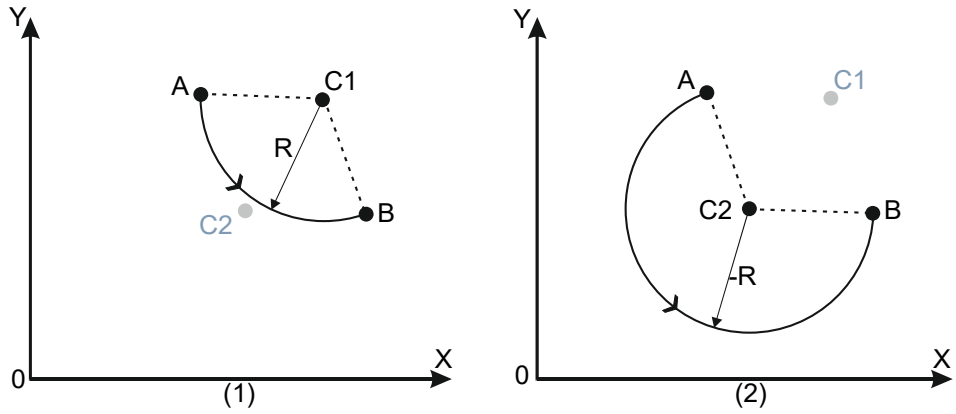
A movement profile is applied on this movement. Refer to §15.4.1.3 for more information.

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 ADVANCED generates an arc of a circle covering less than 180°. If the radius is negative, the ULTIMET ADVANCED 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):

`*ICCWR=0,Xb,Yb,R` (1)
or
`*ICCWR=0,Xb,Yb,-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 joins the badly defined starting point to the corrected one.

Example:

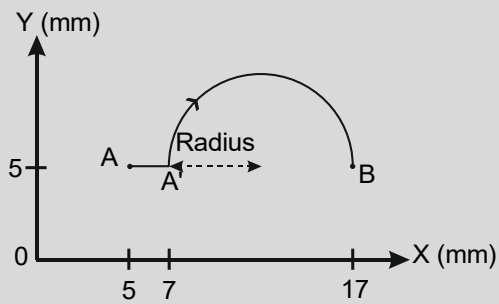
In a XY stage, the current point is A (X=5mm, Y=5mm). The interpolated mode is active and the user position unit is the micron. The following command is sent:

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

This command cannot be executed because the length between the starting and the end point (12mm) is bigger than the diameter of the circle (10mm). The ULTIMET ADVANCED 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=7mm, Y=5mm)
*ICWR = 0, 17000, 5000, 5000; // Clockwise circle to reach point B (X=17mm,
                              // Y=5mm) with a radius of 5mm
```

The circle can be now executed properly.



15.4.1.3 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 on the next segments. The tangential speed is defined according to the unit format described in §15.1.3.

Command	***ITSPD = <P1>, <P2>**		
Description	Sets tangential speed of the selected interpolation group.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group.
<P2>	Tangential speed	From 0 to (2^{31}-1) (*)	Tangential speed.

(*) 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).

From an EDI application:

```
int dsa_ipol_tan_velocity_s(DSA_IPOL_GROUP *igrp, double velocity,
                            long timeout);
```

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

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

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 on the next segments. The 'tangential_acceleration' and 'tangential_deceleration' are defined according to the unit format described in §15.1.3.

Command	***ITACC = <P1>, <P2>** ***ITDEC = <P1>, <P2>**		
Description	Sets tangential acceleration / deceleration of the selected interpolation group.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group
<P2>	Tangential_acceleration or tangential_deceleration	From 0 to (2^{31}-1) (*)	Tangential_acceleration or tangential_deceleration

(*) The maximum value authorized depends on the motor's capabilities.

From an EDI application:

```
int dsa_ipol_tan_acceleration_s(DSA_IPOL_GROUP *igrp, double acc, long timeout);
int dsa_ipol_tan_deceleration_s(DSA_IPOL_GROUP *igrp, double dec, long timeout);
```

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

M	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

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. A jerk time different from 0 will create an S-curve movement. If the jerk time is zero, the profile is a trapezoid one.
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

Command	***ITJRT = <P1>, <P2>**		
Description	Sets tangential jerk time of the selected interpolation group		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group
<P2>	Jerk_time	From 0 to 300 ms	Tangential jerk time

From an EDI application:

```
int dsa_ipol_tan_jerk_time_s(DSA_IPOL_GROUP *igrp, double jerk_time,
                             long timeout);
```

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

M	Comment	Units
***M652**	Tangential jerk time, ipol group 0	ufti
***M672**	Tangential jerk time, ipol group 1	ufti

Example:

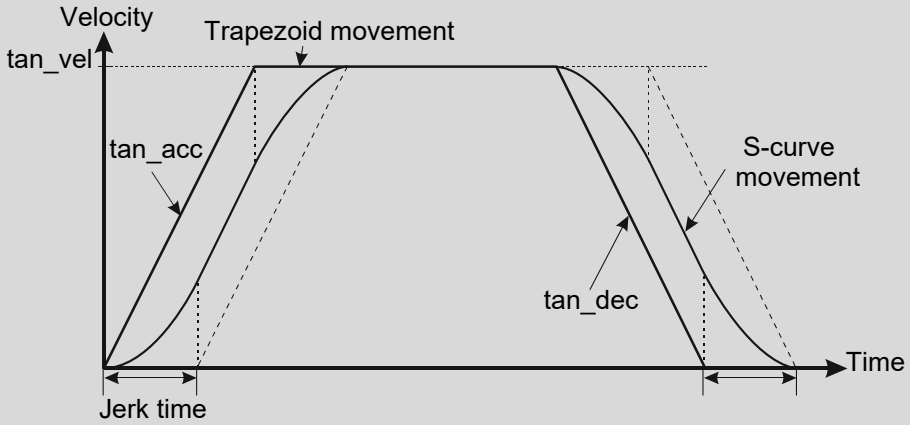
The graphs below illustrate the different commands relating to the trapezoidal and S-curve movement profiles. Both examples are defined with the same tangential speed, the same tangential acceleration and the same tangential deceleration. The length of the movement is the same in both cases. As the movement 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;
```

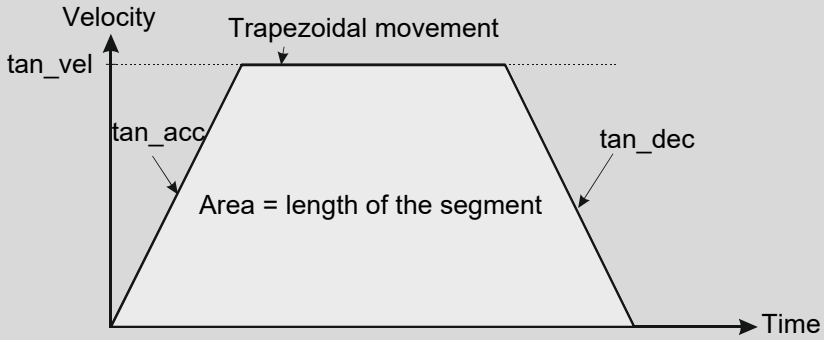
At this point a trapezoid movement is generated (first example)

```
*ITJRT = 0, jerk_time;
```

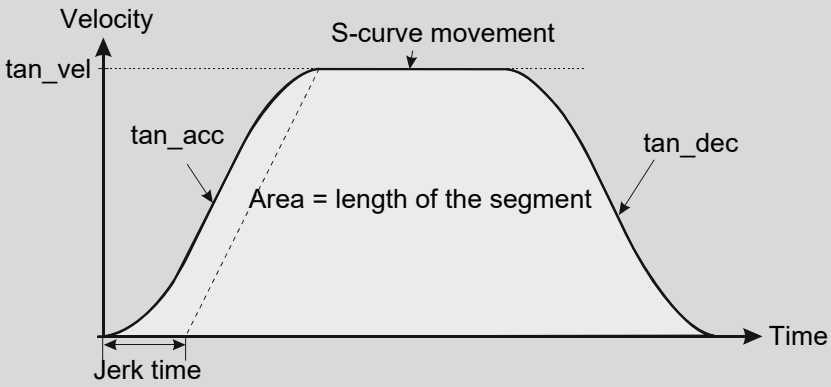
With jerk_time!=0 this defines an S-curve movement (second example)



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



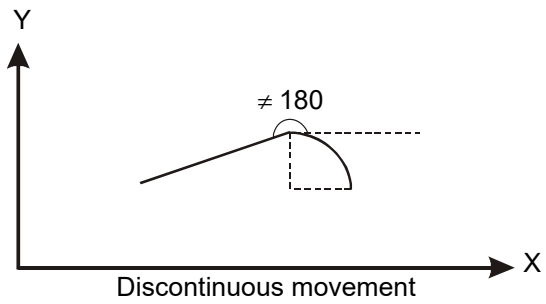
The second movement profile is S-curve (with jerk time).

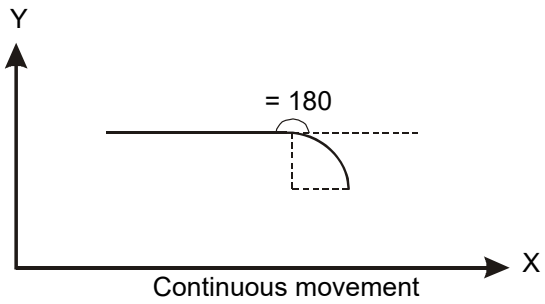


Both movements do not last the same time. This difference is given by the jerk time. In a S-curve movement, the jerk limitation allows smoother transitions on the mechanical system. The fluidity of the movement is a trade-off with the time of execution. The jerk time value is a compromise between the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

15.4.1.4 Concatenated movement commands

During the generation of the trajectory made up with several segments, the default mode of the ULTIMET ADVANCED 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 (same tangents 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 one segment to the next without stopping between them.

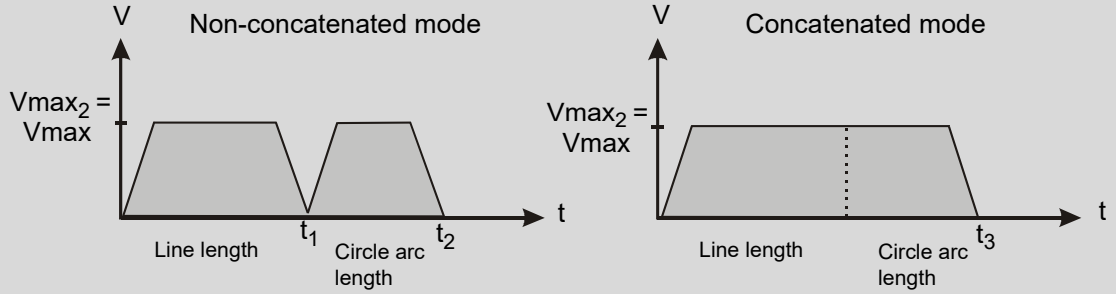




On the right graph above, the linear segment can be concatenated with the arc of a circle and then there is no stop between both segments. If the current tangential speed is not modified, the trapezoidal or S-curve profile applies to the total length of both segments.

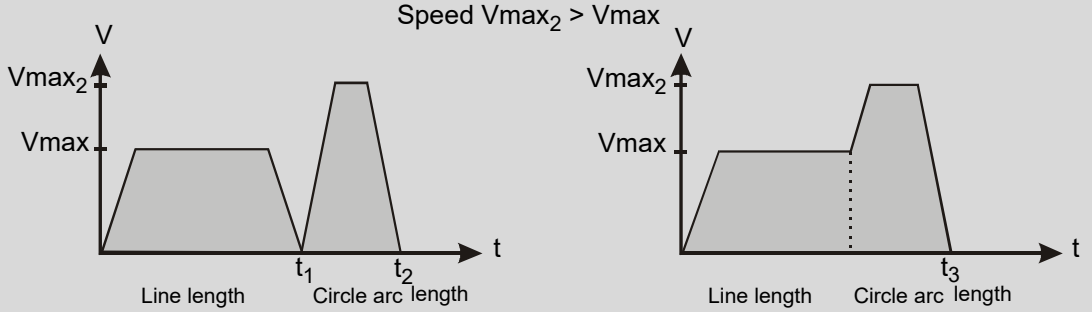
Example:

Speed Vmax = $Vmax_2$



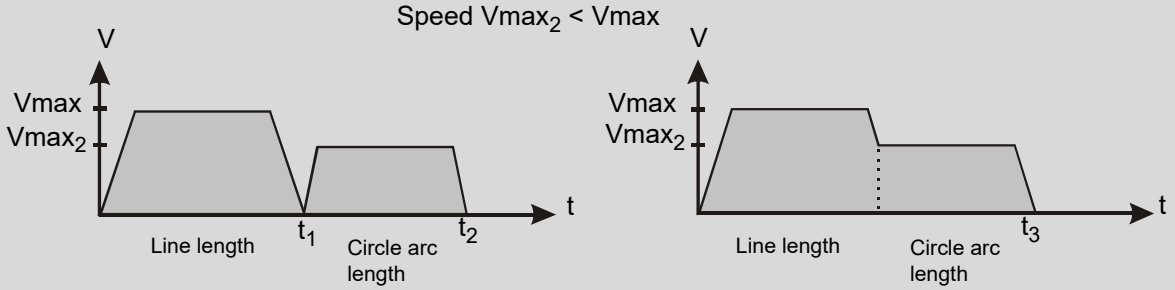
If a speed command is inserted between both segments, the speed profile acts accordingly.
If the new speed is higher than the previous one, the acceleration to reach this speed occurs only at the end of the first segment.

Speed $Vmax_2$ > Vmax



On the other hand, if the new speed is lower, the algorithm induces a deceleration in such a way that the new speed is reached at the beginning of the new segment.

Speed $Vmax_2$ < Vmax



The ICONC and INCONC commands allow the user to manage this mode.

The **ICONC** command (**I**nterpolation **CONC**atenation) activates the concatenated mode during the trajectory and the **INCONC** command (**I**nterpolation **N**o **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 ADVANCED 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 ADVANCED 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.

Command	***ICONC = <P1>** ***INCONC = <P1>**		
Description	Sets tangential jerk time of the selected interpolation group		
Parameter	**Description**	**Value**	**Description**
<P1>	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.

From an EDI application:

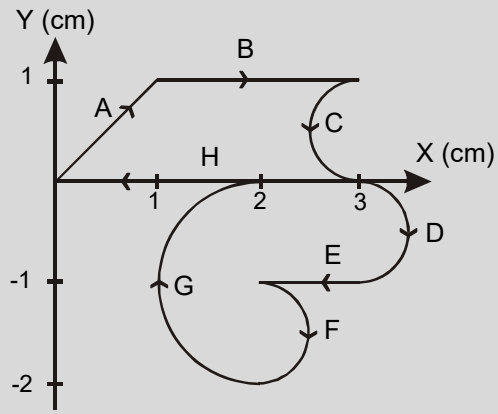
```
int dsa_ipol_begin_concatenation_s(DSA_IPOL_GROUP *igrp, long timeout);
int dsa_ipol_end_concatenation_s(DSA_IPOL_GROUP *igrp, long timeout);
```

Example:

Example with a XY system with two orthogonal and linear axes. With the ISET command, node 5 is associated to the X axis and node 7 to the Y axis. The user unit for position, speed, acceleration and time are microns, microns per second, microns per squared second and microseconds.

```
*ISET = 0, 32L, 128L, 0L, 0L;      // Setting of the interpolated group 0 with
                                   // node 5 for axis X and node 7 for axis Y.
*K522 = 1;                         // Multiplying factor for the position.
*K523 = -6;                        // User unit for the position: here in um.
*K524 = -6;                        // User unit for the speed: here in um/s.
*K525 = -6;                        // User unit for the acceleration: here in
                                   // um/s2.
*K526 = -6;                        // User unit for the time: here in us.
*IBEGIN = 0;                       // Start of the interpolated mode for the
                                   // ULTIMET ADVANCED 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).
*ICONC = 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;                // Segment up to the point (X=0cm, Y=0cm)
```

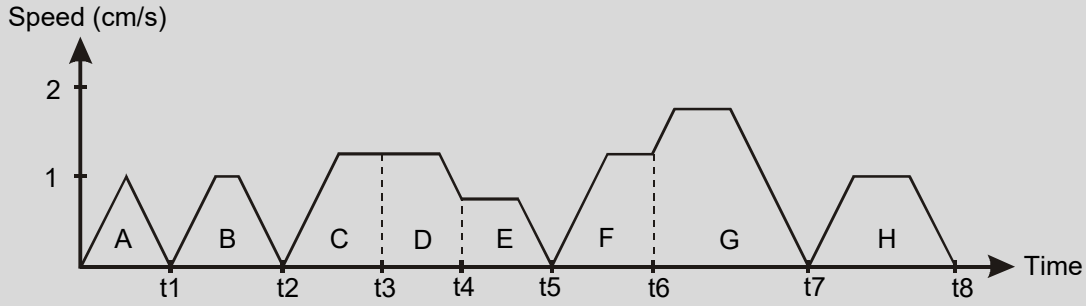
The generated trajectory is then:



In the figure above, 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 as desired. 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 maximum speed of the next segment).

15.4.2 PVT (Position-Velocity-Time) mode

The type of interpolation described in this chapter offers more flexibility (than the G-code based movement) by 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. This ensure a speed continuity but the user has to manage the continuity of the acceleration and to limit the jerk himself when building the trajectory.

The **IPVT** command (**I**nterpolation **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 1 TransnET cycle time (100 µs).
- The minimum time per PVT segment is 1 TransnET cycle time and any value under this time would lead to an error or an uncontrolled trajectory.
- The maximum time per PVT segment is limited by the algorithm to 2.3 seconds and using a higher time segment would lead to an uncontrolled trajectory.

Command	***IPVT = <P1>, <P2>, <P3>, <P4>, <P5>, <P6>, <P7>, <P8>, <P9>, <P10>**		
Description	Adds a PVT (Position-Velocity-Time) move to the trajectory for the selected interpolation group.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group
<P2>	Coord_x	From -distance_max to +distance_max (*)	x coordinates of the target point
<P3>	Coord_y	From -distance_max to +distance_max (*)	y coordinates of the target point
<P4>	Coord_z	From -distance_max to +distance_max (*)	z coordinates of the target point
<P5>	Coord_θ	From -distance_max to +distance_max (*)	θ coordinates of the target point

<P6>	Vel_x	From -vel_max to vel_max (**)	Speed on x of the target point
<P7>	Vel_y	From -vel_max to vel_max (**)	Speed on y of the target point
<P8>	Vel_z	From -vel_max to vel_max (**)	Speed on z of the target point
<P9>	Vel_θ	From -vel_max to vel_max (**)	Speed on θ of the target point
<P10>	Time	From 0 to time_max (***)	Time of the movement

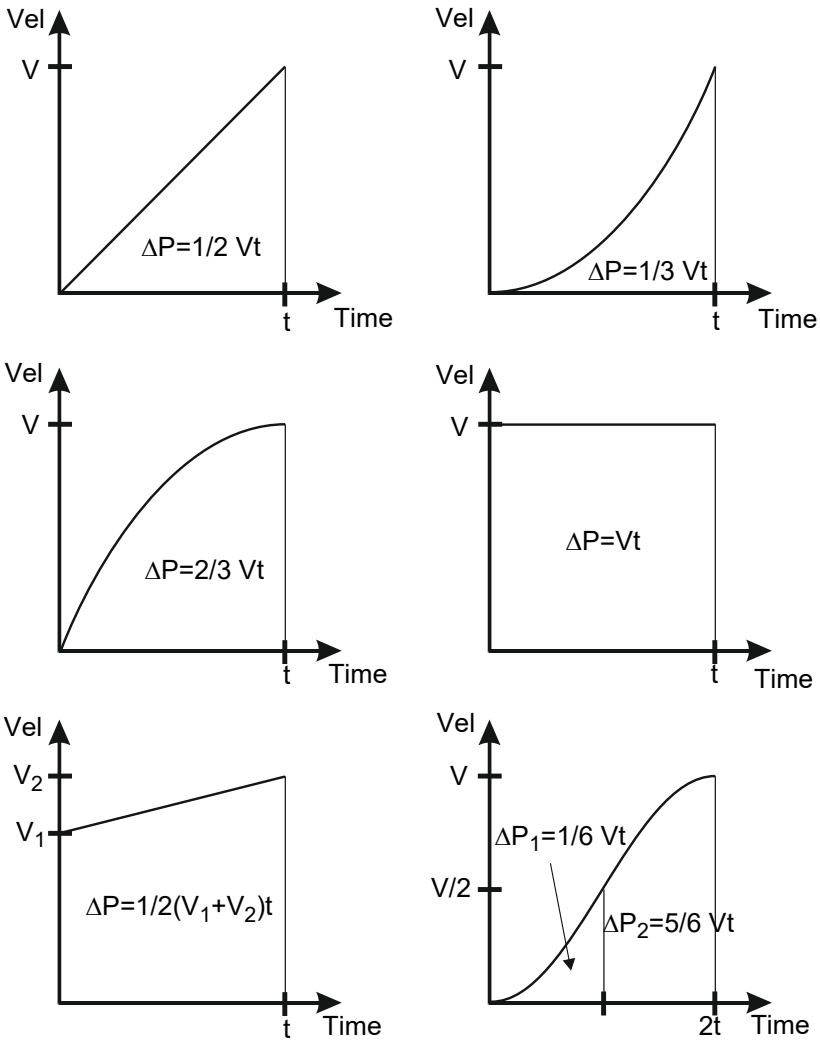
(*): Refer to §15.1.3 for more information.
(**): vel_max is equal to (2^{31}-1)
(***): max. time is 2.3 sec

From an EDI application:

```
int dsa_ipol_pvt_s(DSA_IPOL_GROUP *igrp, DSA_VECTOR *dest, DSA_VECTOR *velocity,
                   double time, long timeout)
```

The relationship between position parameter values and the time value of the *IPVT command has to be as in the following diagrams to obtain the represented profiles:

Profile examples:



Example of a possible PVT trajectory:

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 regiser)). 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 §32. for the list of warnings and to §15.5 for the description of *M519.

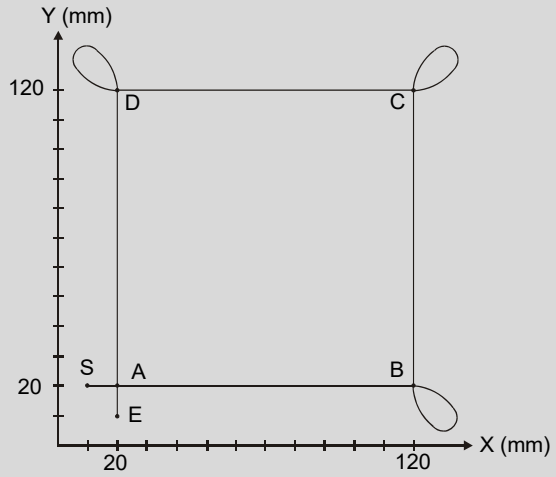
K	Comment	Units
***K546**	Deceleration value for each axis in case of emergency braking in PVT mode (depths 0 = axis X of group 0, 1 = Y of group 0, …, 4 = X of group1, …)	ufai

Example:

In a XY system, a square form is cut into a metal piece. The cutting tool 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 follows: 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
```



Remark: The trajectory loop is outside of the square due to the 3 degree polynomial algorithm.

15.4.2.1 On-the-fly trajectory change

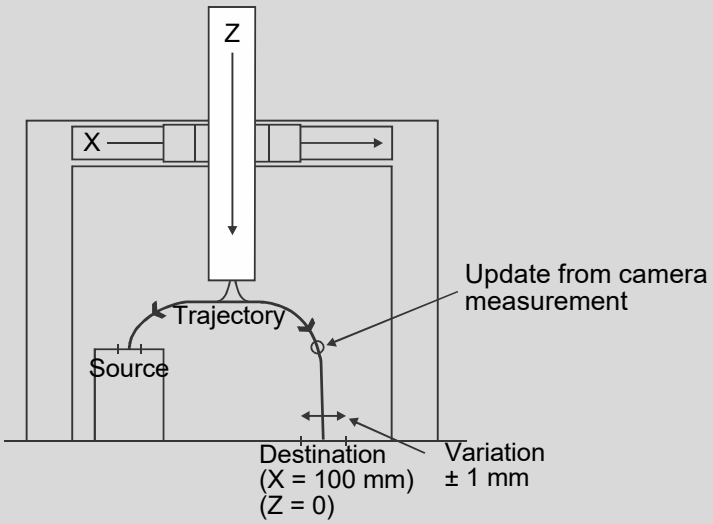
In many applications, the final position of a tool is only known when the movement has already started. For example, in pick and place applications, a camera determines the exact position of a component just before the gripping tool puts it. The trajectory of the gripping tool has to be modified 'on-the-fly' to properly place the component.

The PVT mode can accept on-the-fly modifications of the trajectory. In the above-mentioned example of the IPVT command, the parameters are immediate values. To give the user complete flexibility, the IPVT command can also contain variables.

The PVT mode uses a special set of registers: the Y registers (there are 256 Y registers, each with 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 a Y register can be modified at any time. In the IPVT command, only depth 0 of the Y registers is used.

Example:

In an 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 initially not very precisely known on the X axis (+/- 1 mm).



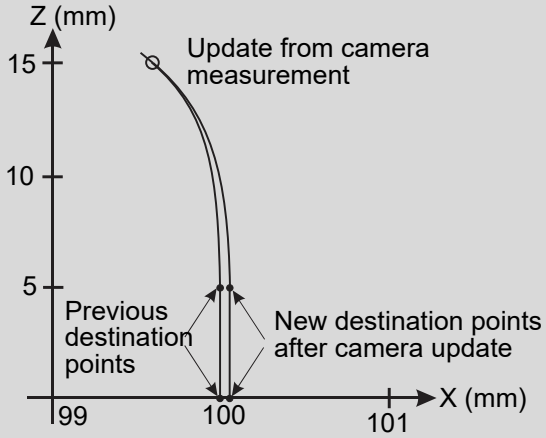
During the execution of the trajectory from the source point to the destination point, a camera helps to determine the exact position of the piece at the destination point. Here is an example of the last commands sent to the ULTIMET ADVANCED to reach the destination point. This destination point has to be reached

vertically to properly put the piece. The user position unit is in microns, user speed unit is in millimeters/second and the user time unit is in milliseconds.

```
*IPVT = 0, Y10, 0, 5000, 0, 0, 10, 0, 0, 15; // With *Y10=100000 corresponding
                                             // to X=100 mm.
*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 above two is running. The X position is updated thanks to the *Y10 variable (*Y10=100100 corresponding to X=100.1 mm).

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 §15.4.2.2.

From an EDI application:

```
int dsa_ipol_pvt_reg_typ_s(DSA_IPOL_GROUP *igrp, DSA_VECTOR*dest,
                           DSA_VECTOR_TYP destTyp, DSA_VECTOR *velocity,
                           DSA_VECTOR_TYP velocityTyp, double time,
                           int timeTyp, long timeout);
```

15.4.2.2 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 index of the first Y register of the block is stored in the parameter ***K560**.

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, are updated by the IPVTUPDATE command. This parameter is a 32 bits word and each bit defines one register.

Command	***IPVTUPDATE = <P1>, <P2>**		
Description	Updates the Y registers for PVT trajectory modifications.		
Parameter	**Description**	**Value**	**Description**
<P1>	Depth	From 0 to 3	Depth where the new value are stored
<P2>	Mask	From 0 to (2^{32}-1)	Mask of the modified Y registers

From an EDI application:

```
int dsa_ipol_pvt_update_s(DSA_IPOL_GROUP *igrp, int depth,
                          dword mask, long timeout);
```

Example:

The previous example (§15.4.2.1) only modifies the X position. In this example, Z position can also be modified with the following commands:

```
*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, 15;
*IPVT = 0, Y10, 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=100 mm, Z=5 mm), the second one is (X=100 mm, Z=0 mm).

Now the following command is sent:

```
*IPVTUPDATE = 1, 0x03;  // Update of the Y registers pointed by *K560 (Y10) and
                        // included in the mask 0x03 = 0b011 i.e. the first
                        // 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=100.1 mm, Z=5.2 mm), the second one is (X=100.1 mm, Z=0 mm).

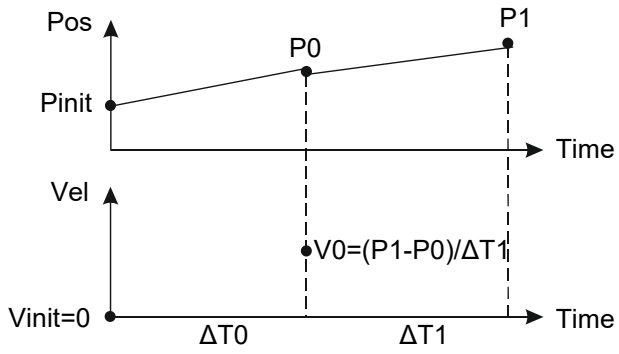
Caution: A parameter change in the PVT mode can induce great modifications of the acceleration in the trajectory generation.

15.4.3 PT (Position-Time) mode

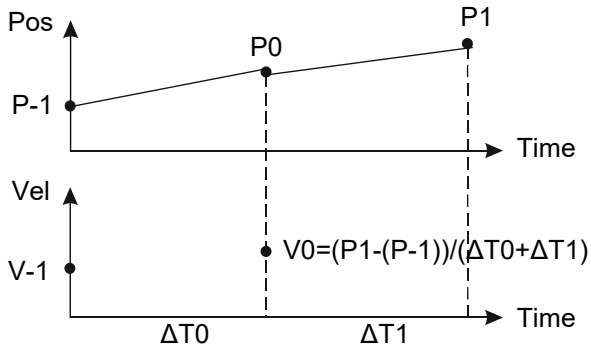
As well as the PVT mode, the type of interpolation described in this chapter offers more flexibility (than the G-code based movement) while leaving the user the definition of the movement profile and the generation of the trajectory. The PT movement describes a segment by giving its arrival point B (the 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 (V0),
      pos_pt(0) is the target point of the current segment (P0),
      pos_pt(1) is the target point of the next segment (P1),

time_pt(1) is the next segment time (ΔT)



- 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 previous segment final velocity V0,
  pos_pt(-1) is the initial point of the current segment P-1,
  pos_pt(1) is the target point of the next segment P1,
  time_pt(1) is the next segment time (ΔT)



- The last segment final velocity is 0

Note: Usually ΔT0 and ΔT1 are the same, the segment time generally being a constant.

The **IPT** command (**I**nterpolation **P**osition-**T**ime) allows the user to execute an interpolated movement on 4 axes in the position-time mode.

As well as for PVT mode, there are some limitations:
- The time parameter of this command must be a multiple of 1 TransnET cycle time (100 µs).
- The minimum time per PT segment is 1 TransnET cycle time 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 an uncontrolled trajectory.

Command	***IPT = <P1>, <P2>, <P3>, <P4>, <P5>, <P6>**		
Description	Adds a PT (Position-Time) move to the trajectory for the selected interpolation group.		
Parameter	**Description**	**Value**	**Description**
<P1>	ipol_group	0 or 1	Interpolated axes group
<P2>	coord_x	From -distance_max to +distance_max (**)	x coordinates of the target point
<P3>	coord_y	From -distance_max to +distance_max (**)	y coordinates of the target point
<P4>	coord_z	From -distance_max to +distance_max (**)	z coordinates of the target point
<P5>	coord_θ	From -distance_max to +distance_max (**)	θ coordinates of the target point
<P6>	time	From 0 to time_max (****)	Time of the movement

(*): Refer to §15.1.3 for more information.
(**): time max is 2.3 seconds.

From an EDI application:

```
int dsa_ipol_pt_s(DSA_IPOL_GROUP *igrp, DSA_VECTOR *dest,
                  double time, long timeout);
```

15.4.4 Braking commands

15.4.4.1 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 (**I**nterpolation **BR**a**K**e) stops the current movement (the braking is done along the trajectory) with a defined deceleration:

- In G-code based mode, this tangential deceleration used is the last one programmed.
- In PVT and PT mode, this deceleration time is defined by the value of parameter *K561.

Remark: After an 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 (§15.4.4.4).

The **ISTP** command (**I**nterpolation **ST**o**P**), 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.

Command	***IBRK = <P1>**		
Description	Stops the selected interpolation group, with the current deceleration value.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group

Command	***ISTP = <P1>**		
Description	Stops the movement of the specified interpolation group with an infinite deceleration.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group

From an EDI application:

```
int dsa_ipol_quick_stop_s(DSA_IPOL_GROUP *igrp, int mode,
                          dword flags, long timeout);
```

IBRK or ISTP will be executed based on `mode` and `flags` parameters. Refer to EDI documentation for more information.

The values corresponding to the IBRK and ICONT commands are calculated according to the time value of parameter ***K561** (refer to §15.4.5.5).

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.

In case the last segment ends with a speed different than 0 the trajectory manager performs an emergency braking with the value stored in parameter ***K546** for the deceleration.

K	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

15.4.4.2 ICONT and ICLRB commands

After a stop with an IBRK or ISTP command, the movement can be either continued or canceled to enable a new movement to start from this point, if there are no errors on the moving axes.

- To continue the movement, the **ICONT** command (**I**nterpolation **CONT**inue), valid for the PVT, PT and G-code based modes, allows the user to resume with the current tangential acceleration for G- code based mode, with *K561 for PVT and PT.
- To cancel the movement, the **ICLRB** command (**I**nterpolation **CL**ea**R** **B**uffer), 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.

Command	***ICONT = <P1>**		
Description	Restarts the previously stopped interpolation group, with the current acceleration value.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group

Command	***ICLRB = <P1>**		
Description	Clears the interpolation command buffer of the selected interpolation group.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group

From an EDI application:

```
int dsa_ipol_continue_s(DSA_IPOL_GROUP *igrp, long timeout);
int dsa_ipol_clear_buffer_s(DSA_IPOL_GROUP *igrp, long timeout);
```

15.4.4.3 ILOCK and IUNLOCK commands

When an interpolation command is sent, it is first processed 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 be able to perform that, 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 pending commands. This command is meaningful and has an effect only if the buffer has been previously stopped by the ILOCK command. As soon as the IUNLOCK command is executed, the movement begins (in one cycle). This command is valid for all modes of interpolation.

Command	***ILOCK = <P1>**		
Description	Locks the selected interpolation group for the selected interpolation group		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group

Command	***IUNLOCK = <P1>**		
Description	Starts the selected interpolation group by unlocking the 'ILOCK' function		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group

From an EDI application:

```
int dsa_ipol_lock_s(DSA_IPOL_GROUP *igrp, long timeout);
int dsa_ipol_unlock_s(DSA_IPOL_GROUP *igrp, long timeout);
```

The monitoring register ***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).

M	Value	Comment
*M531	0 1	The interpolation buffer is not locked The interpolation buffer is locked

15.4.4.4 HLT, HLO and HLB commands

HLT, HLO and HLB commands allow the user to stop the system in case of emergency. These commands are valid for all interpolation modes.

The **HLT** and **HLO** commands
- abort all pending commands,
- stop the interpolated movement with an infinite deceleration and disables the interpolation mode,
- stop the Customer Software Module,
- and stop any SPI cyclic task.

Remark: On ULTIMET ADVANCED, HLT and HLO commands have the same effect. They only coexist for homogeneity with the AccurET controllers (where in addition to stopping the movement like the HLT command, the HLO also powers off the motor).

The **HLB** command
- aborts all pending commands,
- brakes the interpolated movement with a defined deceleration and disables the interpolation mode at the end of the brake movement. This deceleration is defined by the last programmed deceleration in G-code based mode, or by the values *K561 in PVT and PT modes,
- stops the Customer Software Module,
- and stops any SPI cyclic task.

Command	***HLT[.<axis>]**
Description	Stops the CSM threads, stops the interpolated movements with an infinite deceleration and disables the interpolation mode.

Command	***HLO[.<axis>]**
Description	Stops the CSM threads, stops the interpolated movements with an infinite deceleration and disables the interpolation mode.

Command	***HLB[.<axis>]**
Description	Stops the CSM threads, stops the interpolated movements with a defined deceleration and disables the interpolation mode at the end of the movement brake.

From an EDI application, the following command must be used:

```
int dsa_ipol_quick_stop_s(DSA_IPOL_GROUP *igrp, int mode,
                          dword flags, long timeout);
```

HLT, HLB or HLO will be executed, based on the mode and flags parameters. Refer to EDI documentation for more information.

The behavior of the CSM in case of a HLT, HLO and HLB depends on the following conditions:
- The value of *K707 that defines the HLx CSM abort mode,
- The way the CSM is managed (REMOTE_CONTROLLED or not),
- The value of *K144, that defines the mask of the threads that must always keep on running,
- The way the HLx command is transmitted (normal or urgent command).

***K707** defines if the CSM must be abruptly stopped or if the firmware must try to properly end the threads.

K	Name	Value	Comment
***K707**	HLx CSM abort mode	0 (default)	The CSM is immediately stopped, the core#3 is reset and kept in idle state.
		1	The firmware tries to stop the CSM threads if they were started in REMOTE_CONTROLLED mode and are not masked by *K144. After a timeout, the stop is forced by a reset of the core#3 if the HLx command is an urgent command, or an error is raised if it is sent as a normal command.

*K707 = 0 is the safest mode. It should be used by default, except when some threads must be kept alive to protect the machine and the equipment.

*K707 = 1 stops the CSM in a more controlled manner, but only if the REMOTE_CONTROLLED mode is used. If the CSM is not running in REMOTE_CONTROLLED mode, HLx commands have no effect on the CSM if *K707 = 1.

Then if *K707 = 1 and the REMOTE_CONTROLLED is applied, when a HLx command is received, the firmware tries to stop the CSM threads that are not masked by parameter ***K144**.

K	Name	Comment
***K144**	No stopping threads	Mask of the threads that must not be stopped by *HLx commands.

The purpose of *K144 is to be able to have some surveillance threads that continue their job even when a *HLx commands is received.

Finally, the thread end depends if the HLx command is a normal or an urgent command:

- If the HLx command is sent as a “normal command”:
    - The firmware stops the CSM threads not masked and waits for the threads’ termination.
    - The firmware waits for a timeout of 11 seconds. (This time may seem big, but the user’s CSM threads management is not known by the ULTIMET ADVANCED. Please refer to chapter D for CSM threads management and best practice.).
    - When the timeout has expired, an error is raised.

- If the HLx command is sent as an “urgent command”:
    - The firmware stops the not masked CSM threads and waits for threads’ termination.
    - The firmware waits a timeout of 3 seconds.
    - When the timeout has expired, the ULTIMET ADVANCED forces the CSM core to stop. Complete CSM application is stopped and the CSM core remains in idle state until the execution of a new “STARTCSM” command.

15.4.5 Time commands

15.4.5.1 IWTT command

The **IWTT** (**I**nterpolation **W**ai**T** **T**ime) 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 in all interpolation modes.

Command	***IWTT = <P1>, <P2>**		
Description	Waits for a defined time in interpolated mode (selected interpolation group).		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group
<P2>	Waiting_time	From 0 to time_max (*)	Waiting time between movements

(*) 'Time_max' parameter results from the definition of the time unit and the internal resolution of the ULTIMET ADVANCED.

Example:

On the interpolation group 0 with a time in [ms], a waiting command of 0.5 second will be:

```
*IWTT = 0, 500;
```

15.4.5.2 ICAM

ICAM (**I**nterpolation **CAM**), valid for PVT, PT and G-code based modes, is an alias of the parameter ***K717** which expands by a certain percentage the time scale, reducing the kinematic quantities of a movement such as tangential speed, tangential acceleration, tangential deceleration (but not the position to reach). Similarly, the wait time specified with the IWTT command, is increased in the same proportion. The ICAM command is taken into account as soon as the movement is at a stop (both for concatenated and non-concatenated movements). 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:

$$\text{Time_quantity} \cdot \frac{100}{\text{ICAM}}$$

K	Alias	Parameter	<P1> value	Comment	Units
***K717**	ICAM	<P1>	From 1 to 100	Reduction of the movement's characteristics. The depths correspond to the ipol group numbers	[%]

Example:

Consider an XY system with two orthogonal and linear axes with 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 (1 cm,0)
```

Up until here, 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 (1 cm,1 cm)
*WTM = 0;                       // Waiting for the end of the movement
```

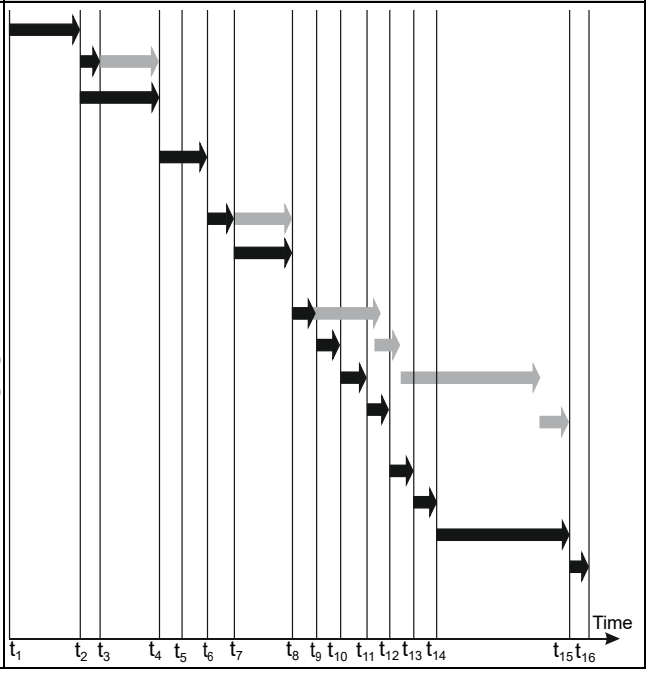
Remark: The length of the first two movements is identical but the execution time is different.

15.4.5.3 *Example of processing time of interpolated commands*

Example 1:

Program	Starting time
dsa_power_on_s(igrp; CMD_TIMEOUT);	t_1 a)
dsa_homing_start_s(igrp, CMD_TIMEOUT);	t_2 a)
dsa_wait_movement_s(igrp, MOVE_TIMEOUT);	t_3 a)
dsa_ipol_begin_s(igrp, DSA_DEF_TIMEOUT);	t_5 a) c)
dsa_ipol_line_s(igrp, VEC (-0.356,-0.517,0,0), CMD_TIMEOUT);	t_4 b)
dsa_ipol_wait_movement_s(ipgr, MOVE_TIMEOUT);	t_5 a)
dsa_ipol_pvt_s(igrp, VEC(-0.355,-0.517,0,0),VEC(0,0,0,0),0.5, CMD_TIMEOUT);	t_6 b) e)
dsa_ipol_pvt_s(igrp, VEC(-0.152,-0.517,0,0),VEC(0.003,0,0,0),0.1, CMD_TIMEOUT);	t_7 b) e)
dsa_ipol_pvt_s(igrp, VEC(0.152,-0.517,0,0),VEC(0.01,0,0,0),1.25, CMD_TIMEOUT);	t_8 b) e)
dsa_ipol_pvt_s(igrp, VEC(0.155,-0.517,0,0),VEC(0,0,0,0),0.01, CMD_TIMEOUT);	t_9 b) e)
dsa_set_digital_output_s(axis_1, 0x1, CMD_TIMEOUT);	t_{10} b)
dsa_set_registers_int32_s(master, DMD_TYP_USER_INT32, 3, 0, 10, CMD_TIMEOUT);	t_{11} b)
dsa_ipol_wait_movement_s(ipgr, MOVE_TIMEOUT);	t_{12} b)
dsa_ipol_end_s(igrp, CMD_TIMEOUT);	t_{13} b)
End of example	t_{14}

Commands processing time



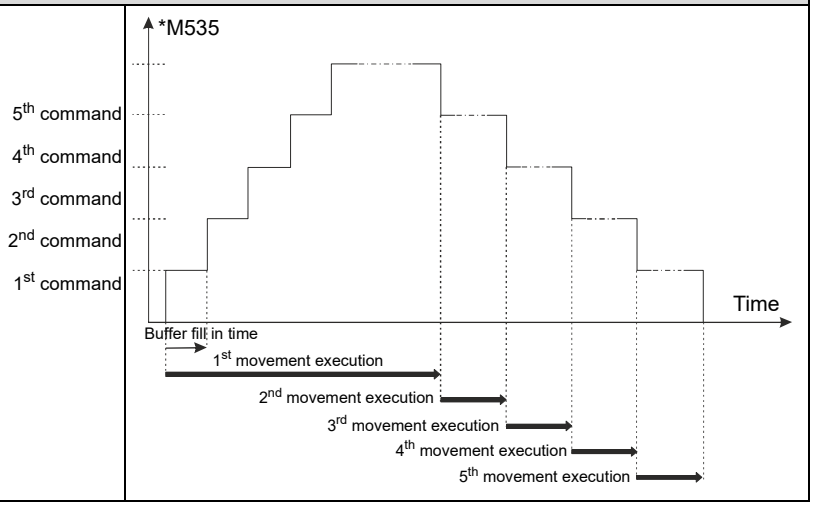
Note: a): command is directly processed as normal command.
b): command is preprocessed and transferred to the interpolation buffer. The movement will start when the previous movement is finished.
Black arrow: command execution, and grey arrow: movement execution
c): `dsa_ipol_begin_s` corresponds to *ISET + *IBEGIN commands
d): `dsa_set_digital_output_s` command starts when all IPVT commands are pre-processed and stored in the interpolation buffer.
e): VEC means DSA_VECTOR.

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:

Program
dsa_ipol_line_s(igrp, VEC (-0.356,-0.517,0,0), CMD_TIMEOUT);
dsa_ipol_pvt_s(igrp,VEC(-0.355,-0.517,0,0),VEC(0,0,0,0), 0.015, CMD_TIMEOUT);
dsa_ipol_pvt_s(igrp,VEC(-0.152,-0.517,0,0),VEC(0.003,0,0,0), 0.1, CMD_TIMEOUT);
dsa_ipol_pvt_s(igrp,VEC(0.152,-0.517,0,0),VEC(0.01,0,0,0), 1.25, CMD_TIMEOUT);
dsa_ipol_pvt_s(igrp,VEC(0.155,-0.517,0,0),VEC(0,0,0,0),0.01, CMD_TIMEOUT);

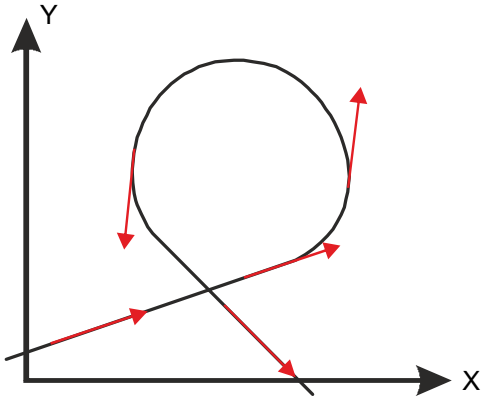
Commands processing time



Remark: The time for a command to be put in the interpolation buffer depends on the UltimET tasks occupation rate. The minimum time is 1.6ms.
VEC means DSA_VECTOR.

15.4.5.4 Tangential speed

The tangential speed (red arrow on the drawing below) is the speed along the trajectory. This speed can be used for any interpolation movement type (G-code based, PVT…) and is accessible when the interpolation mode is active .



The monitoring registers ***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.

Command	***ITANSPDMASK = <P1>, <P2>**		
Description	Defines which axes are considered for calculating the tangential speed.		
Parameter	**Description**	**Value**	**Description**
<P1>	ipol_group	0 or 1	Interpolated axes group
<P2>	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 i.e. 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 is in micrometers.

```
*ITANSPDMASK = 0, 0x7;                // In group 0, axes X, Y and Z are involved
                                      // for the tangential speed.
*ILINE = 0, 10000, 10000, 10000, 0;
*ITANSPDMASK = 0, 0x3;                // In group 0, axes X and Y are involved
                                      // for the 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 register ***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.

ISPDRATE (**I**nterpolation **SP**ee**D RATE**), which is an alias of the parameter ***K530**, allows the user to reduce

the tangential speed on the fly (as opposed to *ITSPD which is executed and taken into account only between segments). The ISPDRATE value is taken into account as soon as it is set. The formula calculating the new value of the tangential speed is:

$$\text{Tangential_speed} \cdot \frac{\text{ISPDRATE}}{100}$$

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

From an EDI application:

```
int dsa_ipol_set_velocity_rate_s(DSA_IPOL_GROUP *igrp,
                                 double rate, long timeout);
```

The values corresponding to ISPDRATE are calculated according to the time value of the parameter *K561 (refer to §15.4.5.5).

15.4.5.5 Speed change time

Some commands provoke a speed change. The time taken to adopt the speed is defined by the parameter ***K561**.

K	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:

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 (1 cm,0)
```

Up until here, 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 §15.4.5.2 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.

15.5 Status

The monitoring register ***M518** indicates if the ULTIMET ADVANCED is executing an interpolated movement or not. Interpolation status information is also provided in the monitoring ***M519**. Each depth of this register corresponds to an interpolation group:

M	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.

15.6 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 in *K540 (refer to §15.6.1) and goes to the next instruction.
Each mark is associated with a number and possibly an action. The number enables the PC or CSM application to identify the mark. The action, programmable via the command's parameters, allows the association of a selective event such as an interruption on the host computer with a particular location on the trajectory.

Command	***ITANSPDMASK = <P1>, <P2>, <P3>, <P4>, <P5>**		
Description	Defines which axes are considered for calculating the tangential speed.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group
<P2>	Mark_number	From 0 to (2^{31}-1)	Number of the mark
<P3>	Action	0-6	Action to be done
<P4>	Parameter1	-	First parameter associated to the action
<P5>	Parameter2	-	Second parameter associated to the action

From an EDI application:

```
int dsa_ipol_mark_2param_s(DSA_IPOL_GROUP *igrp, int number, int operation,
                           int op_param1, int op_param2, long timeout);
```

Remark: It is recommended to use only one *IMARK command between two move commands. In any case, the maximum number of consecutive *IMARK supported is 5.

15.6.1 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 holds the number of the last mark met.

K	Comment
***K540**	Number of the last mark processed by the interpolation group

15.6.2 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
2	Clears the bits of the user status defined by the parameter1
3	Starts a sequence thread (mentioned by parameter2) at the label 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 (Bits 0-15: set, bits 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 30 ns and 0 means no delay)

Operation number	Description
6	Sets / resets several user status bits. The state of the bits is given in parameter1 (Bits 0-15: set, bits 16-31: reset)

The parameters ***K541** allows the user to reserve digital outputs for the marks:

K	Comment
***K541**	DOUTs reserved for the marks

- **WTK command**

The **WTK** (**W**ai**T** for mar**K**) command allows the user to wait for the execution of a mark by the trajectory calculator.

Command	***WTK = <P1>, <P2>**		
Description	Waits for the specified mark <P2> of the defined interpolation group <P1>.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group
<P2>	Mark_number	From 0 to (2^{31}-1)	Number of the mark to wait for

From an EDI application:

```
int dsa_ipol_wait_mark_s(DSA_IPOL_GROUP *igrp, int mark, long timeout);
```

The WTK command constantly checks the parameter ***K540** until it takes the 'mark_number' value. If the parameter *K540 has already the 'mark_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 in all interpolation modes.

15.6.2.1 Set / clear bits of the user status

To set / clear bits of the user status, 2 data items are needed:
- the action value saying if the bits will be set or cleared (defined by <action>=1, =2 resp.).
- 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.

15.6.2.2 Start of a CSM function

To start a CSM function in the ULTIMET ADVANCED, 3 data are needed:

- the action value saying that a jump will be executed (defined by <action>=3).
- the destination label (defined by <parameter1>).
- the CSM thread number used to execute this function code (defined by <parameter2>).

Remark: Please refer to §22.3 for more information about the CSM threads.

Example:

The user wants to send the *IMARK command to jump to the label 100 using the thread number 1 and the mark number 5. To do so, the command will be as follows:

```
*IMARK = 0, 5, 3, 100, 1;
```

15.6.2.3 Setting of an ULTIMET ADVANCED's user variable

To modify a ULTIMET ADVANCED user variable (X), 4 data are needed:
- the action value saying that a X variable will be modified (defined by <action>=4).
- 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:

To modify the value of the variable *X10 to 12345, at depth 2, with the mark number 5, the command would be as follows:

```
*IMARK = 0, 5, 4, 0x0002000A, 12345;
```

15.6.2.4 Set / reset a digital output

To set DOUT2 and reset DOUT1 with mark number 4 and with a delay of 30 µs (1000 times 30 ns step) after the beginning of the next interrupt cycle, the command would be as follows:

```
*K541 = 0x3;
*IMARK = 0, 4, 5, 0x00010002, 1000;
```

15.6.2.5 Set / reset user status bits

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, the command would be as follows:

```
*IMARK = 0, 2, 6, 0x00080002, 0;
```

15.7 Matrix

To execute circular trajectories in 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 switch from one reference frame to another.

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 XY plane of the user reference frame has to be done. The user command describing the circle in the XY plane is then converted to 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. It is possible to correct some mechanical defects by software to increase the accuracy of the system. The mechanical defects linked to the reference frame (non-orthogonal, rotation, scale) can be corrected by using a matrix. The irregular defects as a deviation in the rotation on an axis cannot be corrected by a matrix. In these cases, a correction by Stage Mapping can be used (refer to AccurET Operation & Software for more information).

15.7.1 Translation function

The **IMTRANS** (**I**nterpolation **M**atrix **TRANS**lation) 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.

Command	***IMTRANS = <P1>, <P2>, <P3>, <P4>, <P5>**		
Description	Executes an axes translation of the selected interpolation group.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group
<P2>	Trans_x	From -distance_max to +distance_max (*)	x translation of the reference frame
<P3>	Trans_y	From -distance_max to +distance_max (*)	y translation of the reference frame
<P4>	Trans_z	From -distance_max to +distance_max (*)	z translation of the reference frame
<P5>	Trans_θ	From -distance_max to +distance_max (*)	θ translation of the reference frame

(*) Refer to §15.1.3 for more information.

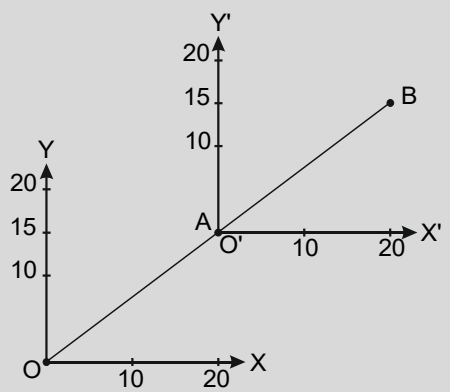
From an EDI application:

```
int dsa_ipol_translate_matrix_s(DSA_IPOL_GROUP *igrp, DSA_VECTOR *trans,
                                long timeout);
int dsa_ipol_translate_matrix_2d_s(DSA_IPOL_GROUP *igrp, double x,
                                   double y, long timeout);
```

Example:

In an 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=20 mm, Y=15 mm).
*IMTRANS = 0, 20000, 15000, 0, 0;  // Translation of the user reference point to
                                   // the point (X=20 mm, Y=15 mm).
*ILINE = 0, 20000, 15000, 0, 0;    // Segment up to point B (X'=20 mm, Y'=15 mm).
```



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.

15.7.2 Scaling function

The **IMSCALE** (**I**nterpolation **M**atrix **SCALE**) 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.

Command	***IMSCALE = <P1>, <P2>, <P3>, <P4>, <P5>**		
Description	Scales the axes of the selected interpolation group.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group
<P2>	Scale_x	From -2^{31} to 2^{31}	x coefficient of the reference frame
<P3>	Scale_y	From -2^{31} to 2^{31}	y coefficient of the reference frame
<P4>	Scale_z	From -2^{31} to 2^{31}	z coefficient of the reference frame
<P5>	Scale_θ	From -2^{31} to 2^{31}	θ coefficient of the reference frame

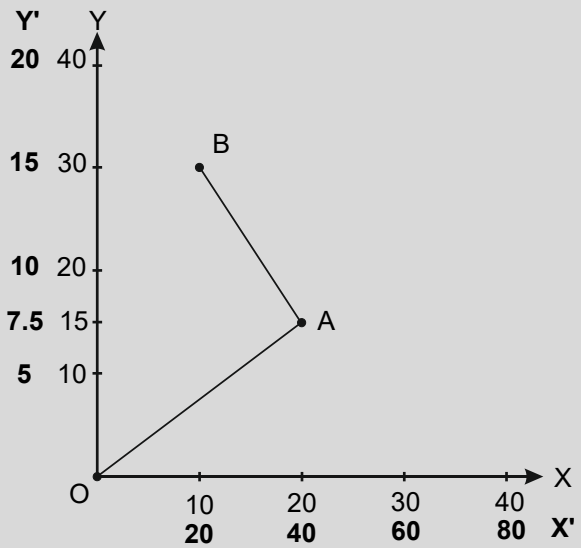
From an EDI application:

```
int dsa_ipol_scale_matrix_s(DSA_IPOL_GROUP *igrp, DSA_VECTOR *scale,
                            long timeout);
int dsa_ipol_scale_matrix_2d_s(DSA_IPOL_GROUP *igrp, double x,
                               double y, long timeout);
```

Example:

In an XY system, let the following commands be executed with the interpolated mode chosen with a tangential speed, a tangential acceleration and a tangential deceleration already set, and the user position unit the micron and the starting point is O (0, 0).

```
*ILINE = 0, 20000, 15000, 0, 0;    // Segment up to point A (X=20 mm, Y=15 mm)
*IMSCALE = 0, 500000, 2000000, 1000000, 1000000;
                                   // Shrink the X axis by half, dilate the Y
                                   // axis by 2.
*ILINE=0,20000,15000,0,0;          // Segment up to point B (X'=20 mm, Y'=15 mm)
```



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

15.7.3 Rotation function

The **IMROT** (**I**nterpolation **M**atrix **ROT**ation) command allows the user to make a rotation of a given plane of the user reference frame.

Command	***IMROT = <P1>, <P2>, <P3>**		
Description	Rotates the axes plane of the selected interpolation group.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group.
<P2>	Plan_rot	0(xy), 1(xz), 2(xθ), 3(yz), 4(yθ), 5(zθ)	Rotation plane.
<P3>	Angle_rot	From -2^{31} to 2^{31}	Rotation angle of the 'plan_rot' plane of the user reference frame.

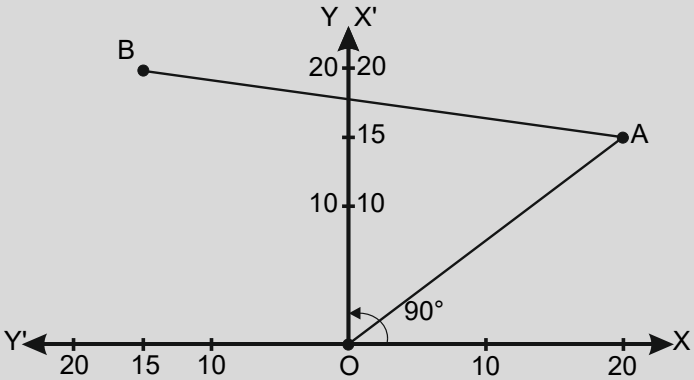
From an EDI application:

```
int dsa_ipol_rotate_matrix_s(DSA_IPOL_GROUP *igrp, int plan,
                             double degree, long timeout);
```

Example:

In an XY system, let the following commands be executed, with the interpolated mode chosen with tangential speed, tangential acceleration and a tangential deceleration already set, and the user position unit the micron and the starting point is O (0, 0).

```
*ILINE = 0, 20000, 15000, 0, 0;  // Segment up to point A (X=20 mm, Y=15 mm)
*IMROT = 0, 0, 90000000;         // 90° rotation of the XY plane
*ILINE = 0, 20000, 15000, 0, 0;  // Segment up to point B (X'=20 mm, Y'=15 mm)
```



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

15.7.4 Shearing function

The **IMSHEAR** (**I**nterpolation **M**atrix **SHEAR**ing) 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.

Command	***IMSHEAR = <P1>, <P2>, <P3>, <P4>, <P5>**		
Description	Shears the defined axis of the selected interpolation group.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group
<P2>	Sheared_axis	0(x), 1(y), 2(z) and 3(θ) (*)	Sheared axis

<P3>	Shear_1	From -2^{31} to 2^{31}	Shearing according to the axis 1
<P4>	Shear_2	From -2^{31} to 2^{31}	Shearing according to the axis 2
<P5>	Shear_3	From -2^{31} to 2^{31}	Shearing according to the axis 3

(*): 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 theta axis.

*IMSHEAR=0,1,100000,0,0;// interpolation group 0, shearing of y and influence on x of 10% of y

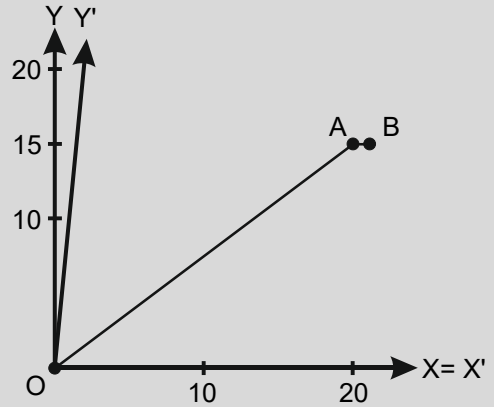
From an EDI application:

```
int dsa_ipol_shear_matrix_s(DSA_IPOL_GROUP *igrp, int sheared_axis,
                            double axis1_shearing, double axis2_shearing,
                            double axis3_shearing, long timeout);
```

Example:

In an XY system, let the following commands be executed, with the interpolated mode chosen with a tangential speed, a tangential acceleration and a tangential deceleration already set, and the user position unit the micron and the starting point is O (0, 0).

```
*ILINE = 0, 20000, 15000, 0, 0; // Segment up to point A (X=20 mm, Y=15 mm)
*IMSHEAR = 0, 1, 100000, 0, 0;  // Sharing 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'=20 mm, Y'=15 mm)
```



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

15.7.5 Matrix restore

The **IMRES** (**I**nterpolation **M**atrix **RES**tore) command allows the user to restore the default matrix of the selected interpolation group. The effect of the 4 other matrix functions is then canceled 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.

Command	***IMRES = <P1>**		
Description	Restore the default matrix of the selected interpolation group.		
Parameter	**Description**	**Value**	**Description**
<P1>	Ipol_group	0 or 1	Interpolated axes group

15.8 User conversion factor

The ULTIMET ADVANCED works at the same resolution as 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 ADVANCED 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 the standard setting, one turn (rotary motor) is equivalent to 1 m (on linear motor).

The user has the possibility to modify this correspondence by choosing any ratio between the different axes (linear or rotary) with parameters ***K580**, ***K581** and ***K582**.

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 for 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).

If the user wants to have 1 turn equivalent to 12.3 mm (e.g. ball screw), the ratio is as follows:

1 turn = 0.0123 meter « 1 meter @ (1/0.0123) turns = (10’000 / 123) turns = 81.300813 turns.

In usual applications, 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.

If X_axis is the rotary motor and Y_axis the linear one:
`*K581:0 = 10000` and `*K582:0 = 123` for the rotary motor.
`*K581:1 = 1` and `*K582:1 = 1` for the linear motor.

Then, 1 meter = 10000 / 123 = *K581 / *K582 of a turn.

16 SPI feature

The SPI communication represents a 4-wire (for the four signals named MISO, MOSI, SS, SCLK) standard data-link level protocol with an RS-485 physical-level implementation.

This SPI bus is available on the ULTIMET ADVANCED. It can be configured as Master or Slave. The available frequency range is 6.25 MHz to 781.25 kHz. The 781.25 kHz frequency is compatible with AccurET controllers when they are used as SPI slaves.

16.1 I/O configuration

To activate the SPI bus, SPI related I/Os have to be configured. Parameters ***K800** to ***K803** have to be set linking SPI signals to the corresponding I/Os.

K	Description	Range
***K800**	Associates the clock signal to a specific IO number	0 to 0x80 (only 1 bit must be set)
***K801**	Associates the MISO signal to a specific IO number	0 to 0x80 (only 1 bit must be set)
***K802**	Associates the MOSI signal to a specific IO number	0 to 0x80 (only 1 bit must be set)
***K803**	Associates the Slave Select signal to a specific IO number	0 to 0x80 (only 1 bit must be set)

16.2 SPI connection /disconnection

The SPI bus is connected/disconnected using the SPICON/SPIDIS commands.

16.2.1 Connection

The **SPICON** (**SPI CON**nection) command requires 5 parameters.

Command	***SPICON = <P1>, <P2>, <P3>, <P4>, <P5>**		
Description	Connect and configure SPI interface		
Parameter	**Description**	**Value**	**Description**
<P1>	MasterOrSlave	0: Slave 1: Master	Master / Slave mode
<P2>	ClockFrequency	2: 25 MHz (not applicable yet) 3: 12.5 MHz (not applicable yet) 4: 6.25 MHz (max. possible value today) 5: 3.125 MHz 6: 1.5625 MHz 7: 0.78125 MHz (compatible with AccurET)	Definition of the clock frequency
<P3>	ClockMode	Mode CPOL CPHA 0 (0,0) 0 0 1 (0,1) 0 1 2 (1,0) 1 0 3 (1,1) 1 1	Combinations of polarity (CPOL) and phases (CPHA)
<P4>	CrcMode	0: No CRC 1: CRC-8 – CCITT (AccurET compatible)	Defines if a CRC is calculated and transmitted with the sent / received data
<P5>	MsbOrLsbFirst	0: MSB first 1: LSB first	Specifies if data is transmitted MSB first or LSB first

The monitoring register ***M263** describes the SPI configuration.

M	Description	Bit#	Comment
*M263	Provides information on SPI configuration	0	0: Slave 1: Master
		1-2	SPI clock mode: Mode CPOL CPHA 0 (0,0) 0 0 1 (0,1) 0 1 2 (1,0) 1 0 3 (1,1) 1 1
		3	Defines the bit order: 0: MSB first; 1: LSB first
		4-6	Defines the clock frequency 0 (000): not applicable 1 (001): not applicable 2 (010): 25 MHz (not applicable yet) 3 (011): 12.5 MHz (not applicable yet) 4 (100): 6.25 MHz 5 (101): 3.125 MHz 6 (110): 1.5625 MHz 7 (111): 0.78125 MHz
		7-15	Reserved for future use
		16	Defines the CRC: 0: CRC disable; 1: CRC enable
		17-19	Defines the CRC size: 0: (000): No CRC 1: (001): CRC-8 2: (010): CRC-16 (-> not implemented) 3: (011): CRC-32 (-> not implemented) 4 to 7: Reserved
		20-21	Reserved for future use
		22-23	Defines the CRC error management: 00: UA in Master or Slave mode and data input, in case of CRC error, puts directly the CRC status bit in error and increase CRC error counter M262; 01: UA in Master only mode and data input, in case of the 1st CRC error, asks for the SPI data message again and in case of a 2nd error puts the CRC status bit in error and increase CRC error counter M262. 10 & 11: Reserved

From an EDI application:

```
int dsa_SPI_connect_s(DSA_MASTER *master, int spi_mode, int clock_frequency,
                      int clock_mode, int crc_mode, int bit_order, long timeout);
```

16.2.2 Disconnection

The SPI disconnection command ***SPIDIS** takes no parameters.

From an EDI application:

```
int dsa_SPI_disconnect_s(DSA_MASTER *master, long timeout);
```

16.2.3 CRC handling

When a CRC is activated, one single CRC is calculated for all transferred bytes. E.g. if 16 bytes are transferred at a time, the CRC is calculated on these 16 bytes and transferred after the 16 bytes.
In the case of cyclic transfer (§16.5) or burst transfer (§16.6), one CRC value is calculated for all the data transferred either in one cycle, or in a burst transfer.
The CRC-8 CITT used is compatible with the SPI implementation of the AccurET controllers.

16.2.3.1 CRC error mode

The parameter ***K811** allows to configure the SPI behavior when a CRC error occurs.

K	Name	Bit#	Comment
***K811**	SPI CRC error management	0	Available when the UA is either in Master or Slave mode. On data reception, with a CRC error, sets the CRC error bit in the UA status word and increments the CRC error counter *M262.
		1	Available only when the UA is in Master mode. On data reception, in case of the 1st CRC error, asks for the SPI data message again and in case of a 2nd error puts the CRC error bit in the UA status word and increments the CRC error counter *M262.

The monitoring register ***M262** gives the SPI CRC error counter.

M	Name	Comment
***M262**	SPI CRC error counter	Gives the SPI CRC error counter.

Remark: This counter is cleared at SPI disconnection or reconnection.

16.3 SPI read-write commands

16.3.1 Commands description

Basic SPI commands allow reading, writing or read-writing data. These commands limit the transmission size to 16 bytes:

16.3.1.1 SPI read command

The **SPIREAD** command allows reading up to 16 bytes from the SPI bus. The number of return parameters will depend on the number of bytes which are requested.

Command	***SPIREAD = <P1>, <P2>[, <P3>[, <P4>[, <P5>]]]**		
Description	Reads <P1> bytes of data on SPI interface		
Parameter	**Description**	**Value**	**Description**
<P1>	nbBytes	1-16	Number of bytes which are requested
<P2>	retValue1	0 to 2^{32}-1	First returned value (bytes 0 to 3)
<P3>	retValue2	0 to 2^{32}-1	2nd returned value (bytes 4 to 7)
<P4>	retValue3	0 to 2^{32}-1	3rd returned value (bytes 8 to 11)
<P5>	retValue4	0 to 2^{32}-1	4th returned value (bytes 12 to 15)

Read values are also accessible through monitoring register ***M261** (4 depths).

M	Name	Comment
***M261**	SPI data received	Gives the data received from SPI communication.

From an EDI application:

```
int dsa_SPI_read_s(DSA_MASTER *master, int nb_bytes,
                   dword read_data[DSA_SPI_MAX_RW_DWORD], long timeout);
```

16.3.1.2 SPI write command

The **SPIWRITE** command allows writing up to 16 bytes to the SPI bus. The number of sent parameters will depend on the number of sent bytes.

Command	*SPIWRITE = <P1>, <P2>[, <P3>[, <P4>[, <P5>]]]		
Description	Write <P1> bytes of data on SPI interface.		
Parameter	**Description**	**Value**	**Description**
<P1>	NbBytes	1-16	Number of bytes which are sent
<P2>	SentValue1	0 to 2^{32}-1	First sent value (bytes 0 to 3)
<P3>	SentValue2	0 to 2^{32}-1	2nd sent value (bytes 4 to 7)
<P4>	SentValue3	0 to 2^{32}-1	3rd sent value (bytes 8 to 11)
<P5>	SentValue4	0 to 2^{32}-1	4th sent value (bytes 12 to 15)

From an EDI application:

```
int dsa_SPI_write_s(DSA_MASTER *master, int nb_bytes,
                    dword write_data[DSA_SPI_MAX_RW_DWORD], long timeout);
```

16.3.1.3 SPI read-write command

The **SPIRW** command allows writing and reading up to 16 bytes with the SPI bus. The number of return parameters will depend on the number of bytes which are requested.

Command	*SPIRW = <P1>, <P2>, <P3>, <P4>, <P5>, <P6>[, <P7>[, <P8>[, <P9>]]]		
Description	Write and read <P1> bytes of data on SPI interface.		
Parameter	**Description**	**Value**	**Description**
<P1>	NbBytes	1-16	Number of bytes which are sent / requested
<P2>	SentValue1	0 to 2^{32}-1	First sent value (bytes 0 to 3)
<P3>	SentValue2	0 to 2^{32}-1	2nd sent value (bytes 4 to 7)
<P4>	SentValue3	0 to 2^{32}-1	3rd sent value (bytes 8 to 11)
<P5>	SentValue4	0 to 2^{32}-1	4th sent value (bytes 12 to 15)
<P6>	RetValue1	0 to 2^{32}-1	First returned value (bytes 0 to 3)
<P7>	RetValue2	0 to 2^{32}-1	2nd returned value (bytes 4 to 7)
<P8>	RetValue3	0 to 2^{32}-1	3rd returned value (bytes 8 to 11)
<P9>	RetValue4	0 to 2^{32}-1	4th returned value (bytes 12 to 15)

From an EDI application:

```
int dsa_SPI_read_write_s(DSA_MASTER *master, int nb_bytes,
                         dword wdata[DSA_SPI_MAX_RW_DWORD],
                         dword rdata[DSA_SPI_MAX_RW_DWORD], long timeout);
```

16.3.2 Abort command

The **SPIABORT** command allows aborting a SPIREAD, SPIWRITE or SPIRW. This is mostly helpful when the SPI is configured as a slave. Read-Write commands are blocking commands. Thus if the SPI master does not start a transmission, the SPI slave waits for ever.
This command has to be sent as an urgent command!

Command	!*SPIABORT
Description	Aborts SPI transfer if any.

This command does not disconnect the SPI, it only aborts the current read-write command.

16.4 Transmission status

The current SPI transmission status can be observed with the monitoring register ***M260**:

M	Name	Bit#	Comment
***M260**	SPI status	0	Defines the output buffer status: 0: empty (you can write some new data to be sent) 1: full, waiting for the transmission to start (works only for Slave mode, a start of communication will be triggered if it is a Master). If data are written in the buffer while in this state, previous data will be overwritten. Check bit#1 before interpreting this one.
		1	This bit indicates the end of the transmission, meaning new data are ready to be read. This bit is cleared when writing new data, so in case the user is waiting for new data, he must wait for this bit to be equal to 1 (and not test bit#0) before writing the next data to be sent. 0: Transmission in progress 1: New data is ready to be read Bit is set at the end of transmission, cleared by sending new data
		2-11	Reserved
		12	Defines the CRC error: 0: CRC Ok; 1: CRC error
		16	Set if the last SPI command was aborted

16.5 Cyclic transfer mode

16.5.1 Feature

The “Cyclic transfer mode” offers the possibility for the user to activate a cyclic task in ULTIMET ADVANCED for cyclically transferring data over SPI with any real-time application interface (e.g. Customer Software Module). At each “user defined” cycle, a user defined handler is called for the user to execute a cyclic action based on received SPI data.

Remark: The feature is only available through a programming interface and there are no “commands” associated with it.

Before activating this “user defined” cyclic behavior, the user has to configure the following information:
- Data exchange mode: READ (0x1) – WRITE (0x2) – RW (Read and write, 0x3).
- Exchanged data size: size in bytes.
- Initial application data to be sent to the SPI device: this data will be transferred during the first SPI exchange.

```
int dsa_config_cyclic_transfer_s(DSA_MASTER *master, dword r_w_mode, int size,
                                 byte *write_buffer, long timeout);
```

The start of this cyclic transfer mode is done using:

```
int dsa_start_cyclic_transfer_handler(DSA_MASTER *obj, int rate,
                                      DSA_CYCLIC_TRANSFER_HANDLER handler,
                                      void *user_param);
```

With:
- Rate: multiple of the TransnET rate for defining the SPI transfer rate. Minimum value: 2.
- Handler: will contain the user code (here-below, a handler example §16.5.4).
- User_param: any user structure for exchanging data between the handler and the application.

Two functions are available to read and write transferred data in the handler:

```
int dsa_write_transfer_buffer(DSA_MASTER *obj, unsigned int size,
                              byte *write_buffer);
int dsa_read_transfer_buffer(DSA_MASTER *obj, unsigned int size,
                             byte *read_buffer);
```

The first time the handler is called, there no data to read. Thus the data “made available” by the first call to `dsa_read_transfer_buffer` has to be discarded.

Refer to §16.5.4 for what a cyclic handler may contain.

The cyclic mode can be stopped using:

```
int dsa_stop_cyclic_transfer_handler(DSA_MASTER *obj);
```

16.5.2 Amount of bytes that can be transferred

The ULTIMET ADVANCED can theoretically transfer up to 9182 bytes a during one transfer cycle. Practically the amount of data that can be transferred is limited by:

- The SPI frequency.
- The cyclic rate defined by the user.

For the user to calculate which amount of data can be transferred in one transfer cycle, the following rule is given:

- SPI data can be transferred during a maximum of 50% of the high priority interrupt (HPI) cycle time. Default configuration of the HPI cycle time is 100 µs. Thus SPI data transfer time per 100 µs is 50 µs.
- Maximum amount of bytes being transferred in one HPI cycle time is 128 bytes (inherent to SPI design).
- Algorithm for calculating the amount of data that can be transferred in one transfer cycle:

```
nbBytesPerUAPeriod = floor(50us * spiFrequency);
nbBytesPerUAPeriod = MIN(nbBytesPerUAPeriod, 128bytes);
requiredNbUaPeriods = ceil(NbOfBytesToTransfer / nbBytesPerUAPeriod);
if (requiredNbUaPeriods+1 > cyclicRate)
    // Transfer is not allowed
else
    // Transfer is allowed
```

16.5.3 Watchdog and utility functions

A watchdog can be set using EDI functions. This watchdog is valid for real-time application interfaces only. This watchdog will generate an error if the cyclic transfer handler was not started when the watchdog time has elapsed.

```
int dsa_set_cyclic_transfer_handler_watchdog(DSA_MASTER *master, ebool enable);
```

The watchdog is either enabled or disabled. Enabling the watchdog must be done prior to starting the handler. If the time between two calls to the handler is greater than expected (with margin due to firmware processing time jitter), the following error is raised:

Error 100002: Cyclic handler watchdog error

To know if the cyclic transfer handler is active or not, the following function can be called:

```
int dsa_get_cyclic_transfer_handler_activity(DSA_MASTER *master, ebool *active);
```

In order to access information about cyclic handler processing time, the following function is available, which provides time information (min / max / mean) between the time when cyclic interrupt is raised and the end of the handler.

```
int dsa_get_cyclic_transfer_handler_times(DSA_MASTER *master, double *max_time,
                                          double *min_time, double *mean_time);
```

These times can be reset using the following function:

```
int dsa_reset_cyclic_transfer_handler_times(DSA_MASTER *master);
```

16.5.4 Cyclic handler example

Handler example:

```
// Must be kept! Must be set to true before starting cyclic handler
bool isFirstCyclicHandlerCycle = true;

static void DSA_CALLBACK _cyclicHandler(DSA_MASTER *ultimet, int size,
                                        void* user)
{
```

```
    static int counter = 0;
    int err = 0;

    // Must be kept! At first cycle, no available read data
    if (isFirstCyclicHandlerCycle == false)
    {
        if (cyclicRwMode & READ_MODE)
        {
            if ((err = dsa_read_transfer_buffer(ultimet, size, readBuffer)))
            {
                DSA_EXT_DIAG(err, ultimet);
                goto _error;
            }
        }

        // User code
        // ...
    }
    else
    {
        // User code
        // ...
    }

    if (cyclicRwMode & WRITE_MODE)
    {
        // writeBuffer content to be defined by the user
        if ((err = dsa_write_transfer_buffer(ultimet, size, writeBuffer)))
        {
            DSA_EXT_DIAG(err, ultimet);
            goto _error;
        }
    }
    // Must be kept!
    if (isFirstCyclicHandlerCycle)
    {
        // Must be kept!
        isFirstCyclicHandlerCycle = false;
    }
    counter++;

_error:
    return;
}
```

Start / stop handler example:

```
// Definitions and variables declaration
#define MAX_NB_TRANSFERED_BYTES 128 // max number of bytes that may be
                                    // transfered in this cyclic task

byte initialWriteBuffer[MAX_NB_TRANSFERED_BYTES] = { 0 };
int iDataSize; // number of bytes that will be transferred
int iCyclicTimeRate;

...
// cyclic handler start / stop

// variables configuration
iDataSize = 16; // 16 bytes will be cyclically transferred
iCyclicTimeRate = 2; // these bytes will be transferred each two TransnET cycles

for (i=0; i<iDataSize; i++)
{
    // this init value must be defined by the user and depends on his
    // communication protocol
    initialWriteBuffer[i] = APP_DEFINED_INIT_VALUE;
}
// Watchdog activation
// - it will react on missed handler calls
if ((err = dsa_set_cyclic_transfer_handler_watchdog(ultimet, TRUE)))
```

```
{
    DSA_EXT_DIAG(err, ultimet);
    goto _error;
}

// Configuration and start of the cyclic task
// - this example configures a read-write cyclic transfer of "iDataSize" bytes
// - initialWriteBuffer contains data that will be transferred during the first
// cycle
if ((err = dsa_config_cyclic_transfer_s(ultimet, DSA_TRANSFER_READ_WRITE_MODE,
                                        iDataSize, initialWriteBuffer,
                                        DSA_DEF_TIMEOUT)))
{
    DSA_EXT_DIAG(err, ultimet);
    goto _error;
}

// Must be kept! Must be set to true before starting cyclic handler
isFirstCyclicHandlerCycle = true;

// start of the cyclic task, executed each "iCyclicTimeRate" TransnET cycles
// - each time this task wakes up, it calls the "_cyclicHandler" function
if ((err = dsa_start_cyclic_transfer_handler(ultimet, iCyclicTimeRate,
                                             _cyclicHandler, NULL)))
{
    DSA_EXT_DIAG(err, ultimet);
    goto _error;
}

// Application execution
...

// stop cyclic task
if ((err = dsa_stop_cyclic_transfer_handler(ultimet)))
{
    DSA_EXT_DIAG(err, ultimet);
    goto _error;
}

// disable cyclic handler watchdog
if ((err = dsa_set_cyclic_transfer_handler_watchdog(ultimet, FALSE)))
{
    DSA_EXT_DIAG(err, ultimet);
    goto _error;
}
```

16.6 Burst transfer mode

The “Burst transfer mode” offers the possibility for the user to transfer through the SPI bus a big amount of data from any real-time application interface (e.g. Customer Software Module). This burst mode can transfer up to 16384 bytes at a time.

The following EDI function can be used:

```
int dsa_start_burst_transfer_s(DSA_MASTER *master, dword rw_mode,
                               unsigned int size, byte *write_buffer,
                               byte *read_buffer, long timeout);
```

17 Feature licenses

17.1 Management

Some ULTIMET ADVANCED features need to be activated using a license certificate. This certificate is linked to the serial number of the product and therefore can only be used on the ULTIMET ADVANCED for which it has been generated (the error 4802 will be raised in case of incompatible serial number).

The list of available licensed features depends on the firmware version. For example, from firmware version 3.50A the QUIET ESM adds some additional licensed features to the ULTIMET ADVANCED. Refer to §17.2 for the complete list of features under license.

If there is no license file on the ULTIMET ADVANCED (or if it is not valid, e.g. the serial number it refers to does not correspond to the serial number of the ULTIMET ADVANCED) the features under license can still be used for a period of time called a “grace period”. The duration of the grace period is 15 days and starts when it is detected that a feature under license has been enabled without a valid license file. A permanent warning (*M66 = 8) is raised during this grace period.

The goal of the grace period is twofold:

- Consider the event of an ULTIMET ADVANCED requiring a hardware to be replaced by another. The new board may not have a license file for it and so the user has 15 days to get the licenses without impacting the production flow.
- It allows to test the performances and get a feel of a given feature under license.

The certificate file must be downloaded to the controller with the QCT tool. Refer to the “QCT User’s Manual” for more details about getting and downloading a license file. In case of a bad license file format, the error 4800 will be raised. In addition, if a new license file is downloaded and the latter is not valid, the warning *M66 = 9 will be raised and the previous file is kept.

The certificate is then taken into account at the following reboot of the ULTIMET ADVANCED. To not forget this point, the permanent error 4805 is raised when the license file is updated.

A set of errors and warnings are defined for the license management. The error codes are in the range 4800-4811. Please refer to §32. and §33. for a complete description and troubleshooting hints.

17.2 Configuration

The features under license are identified by a bit number in the registers provided to manage the licenses. These registers contain 64 bits and are therefore of KL or ML type.

On the day this manual is released, the following features under license are available:

Feature Name	Description	License Management Bit #
“DFC Basic in plane 3DOF”	QUIET ESM effective only in plane (X, Y, Rz)	10
“DFC out of plane +3DOF”	QUIET ESM effective in all directions (X, Y, Z, Rx, Ry, Rz)	11
“DFC High performance”	QUIET ESM feature enabling the maximum dampening of the system.	12

Refer to the “QCT User’s Manual” for requesting and downloading a QUIET ESM license.

The list of available features under license in the current firmware version is shown in **ML255**:

M	Name	Comment
***ML255**	Licensable features mask	Bit set: feature subject to license. Refer to the previous table to get the bit definition.

The features for which a license has been bought are shown in **ML256**:

M	Name	Comment
***ML256**	Licensed features mask	Bit set: feature for which a license has been bought.

The features under license to activate can be selected with **KL257**:

M	Name	Comment
***KL257**	Selected licensed features mask	Bit set: feature to activate. By default, all the features are active (i.e default value is 0xFFFF'FFFF'FFFF'FFFF).

A means to deactivate a licensed feature is provided for two reasons:

- When a faulty ULTIMET ADVANCED needs to be replaced by another, all features will be available during the grace period if no license file is available. This might change the behavior of the system in an unwanted way. The possibility to deactivate some grace period granted features allows to ensure the same behavior as with the initial configuration.
- It allows to test the effect with or without a given feature.

Finally, **ML257** reflects the features that are really active: those for which a license is valid (bought or grace period) as shown in *ML256 and selected by *KL257.

M	Name	Comment
***ML257**	Grace period	Bit set: feature activate.

Two more monitoring concern the grace period:

M	Name	Comment
***M255**	Active licensed features mask	Bit set: the ULTIMET ADVANCED is in a grace period
***M256**	Grace period down counter	Number of minutes remaining in the grace period.

17.3 Example

- **Certificate example**

Let us consider as example a product and firmware which comprises three licensable features A, B and C. A firmware license is purchased and downloaded to the product to authorize features A and C, as illustrated in the table below. By default, these features are active.

	A	B	C	Register bit mask
Existing features (*ML255)	✓	✓	✓	$(111)_2$
Authorized features (*ML256)	✓	-	✓	$(101)_2$
Active features (*ML257)	✓	-	✓	$(101)_2$

- **Feature activation example**

As described in the previous chapter, it is possible to select the features to be activated with *KL257. To continue with the previous example illustrated in the table below, only feature A is activated among the authorized features.

	A	B	C	Register bit mask
Existing features (*ML255)	✓	✓	✓	$(111)_2$
Authorized features (*ML256)	✓	-	✓	$(101)_2$
Selected features to activate (*KL257)	✓	-	-	$(001)_2$
Active features (*ML257)	✓	-	-	$(001)_2$

The ULTIMET ADVANCED will raise the error 4804 or 4811 when attempting to activate an unauthorized feature (B in this example).

- **Grace period example**

If a product is used into which no certificate has been downloaded, all features can be temporarily activated taking advantage of the grace period. However, a countdown starts and the remaining grace period can be displayed in *M256.

	A	B	C	Register bit mask
Existing features (*ML255)	✓	✓	✓	$(111)_2$
Authorized features (*ML256)	✓	✓	✓	$(111)_2$
Selected features to activate (*KL257)	✓	✓	-	$(011)_2$
Active features (*ML257)	✓	✓	-	$(011)_2$

The product will raise the error 4810 when the grace period has expired.

THIS PAGE IS INTENTIONALLY LEFT BLANK

Chapter D: Programming

18 Customer Software Module

18.1 Prerequisites

A Customer Software Module (CSM) is a FreeRTOS application based on the EDI library. It is, therefore, recommended to have prior experience on the following subjects:

- Embedded software development using the Xilinx SDK environment (IDE based on Eclipse).
- ETEL application development using EDI (ETEL trainings are available).
- C programming. A CSM can only be programmed in C.

18.2 Toolchains

The Xilinx SDK environment is necessary for the development of a CSM. The customer must use the CSM template provided by ETEL to ensure correct operation of the CSM. The template contains common useful features to support developers creating their embedded application. The CSM is compiled and linked with the EDI library ported to the FreeRTOS operating system. This process generates two binary files:

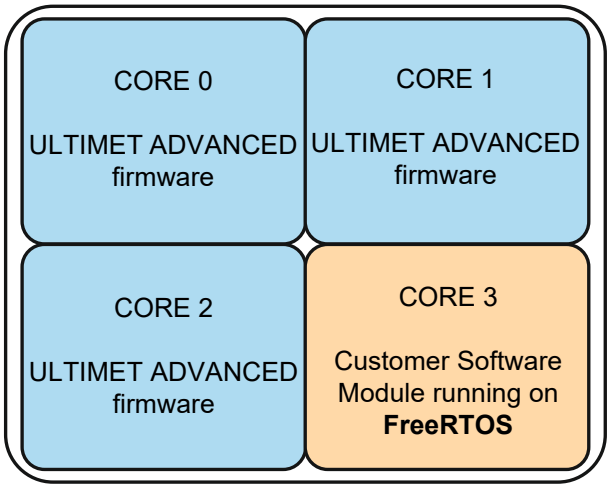
- csm.elf: used during the development phase since it can be launched, debugged and tested using the Xilinx SDK environment.
- csm.bin: used in the final production environment, when the CSM is loaded and started from a host application (it can be **ComET** or a user application running on the host PC).

In addition, a CSM can also be built and run as an application on Windows OS since the same EDI interface exists for both systems. This mode is particularly useful for troubleshooting problems during the development phase. The Windows-built CSM being able to connect to the ULTIMET ADVANCED via a TCP/IP or PCIe interface instead of the shared memory port. Note that real-time functionalities such as RTV and SPI cannot be used when connected via the TCP/IP interface.

18.3 General principle

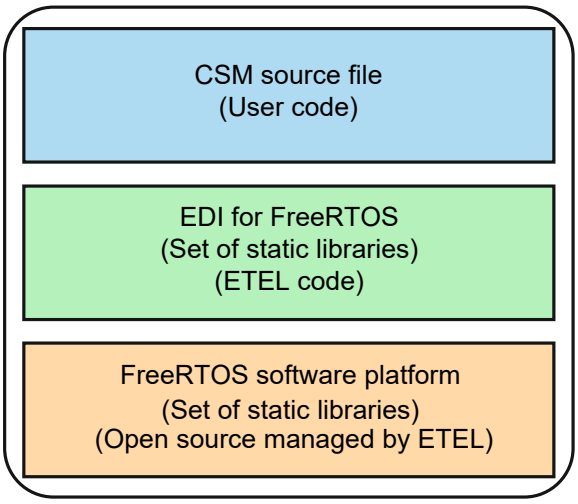
The ULTIMET ADVANCED runs on a quad-core ARM processor, managed by 2 operating systems:

- Three cores are dedicated to ULTIMET ADVANCED firmware. These 3 cores are managed by a single operating system (OS).
- One core is reserved for the Customer Software Module and runs using the FreeRTOS real-time operating system.



18.4 FreeRTOS introduction

FreeRTOS is a free real-time operating system particularly suitable for embedded devices. It is distributed under the MIT License. Please refer to the www.freertos.org website for a complete introduction and documentation.
ETEL delivers a specific version of FreeRTOS as a set of **static** libraries and include files which must be linked with the CSM. A CSM is then composed as shown below:



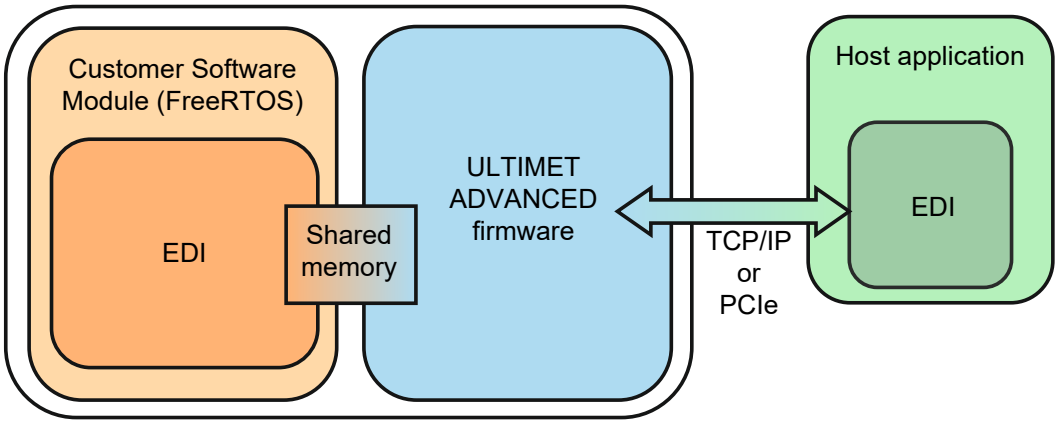
As all provided libraries are static, the user must recompile and relink its application when ETEL provides a new version of EDI or FreeRTOS.

FreeRTOS operating system has the following characteristics:
- Non-SMP multitasking real-time system.
- No virtual memory.
- All tasks are running in the same memory space (only threads, no processes).
- There is no way to spy the OS state (running tasks, used memory, …).

For this 4th core, the overall available data memory is 255 MB of RAM. This memory has to be shared between the CSM binary file, the FreeRTOS heap and the tasks' stack, considering that:
- Each task (or thread) has a stack of 4096 elements of 64bits.
- The reserved heap size is 128 MB.

18.5 Firmware – CSM communication

The CSM communicates with the ULTIMET ADVANCED firmware (and connected AccurET controllers) through the shared memory port. As a normal EDI application, the CSM opens the communication to all axes it requires. Once the communication is established, the CSM can access the axes using any EDI function (apart from the few exceptions presented in §24). When finished, the CSM simply closes the communication.

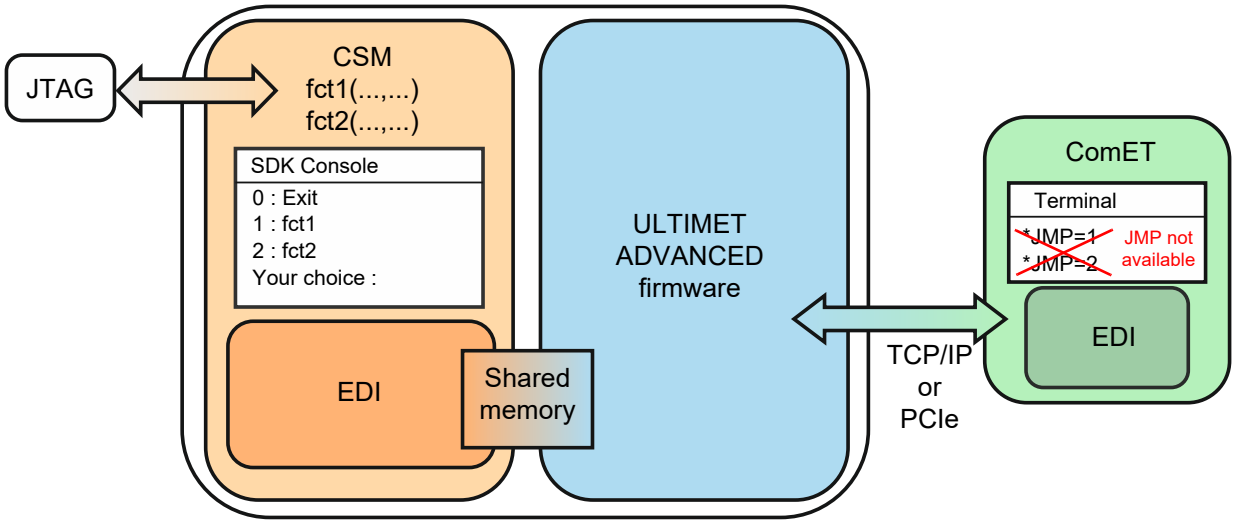


From the user's point of view, there are similarities between a CSM and the UltimET Light sequences. The CSM can also be controlled by a host application using the ***JMP** or ***END** commands.
To support this behavior, the CSM must be developed as a "server" giving access to high-level functions. These high-level functions can be called in two distinct modes:
- During development phase, using the Xilinx SDK console, the CSM user interface displays a menu with a list of supported high-level functions. The user can then choose using the keyboard which specific function to execute. In this mode, it is not possible to communicate with the CSM via an application running on a host PC.

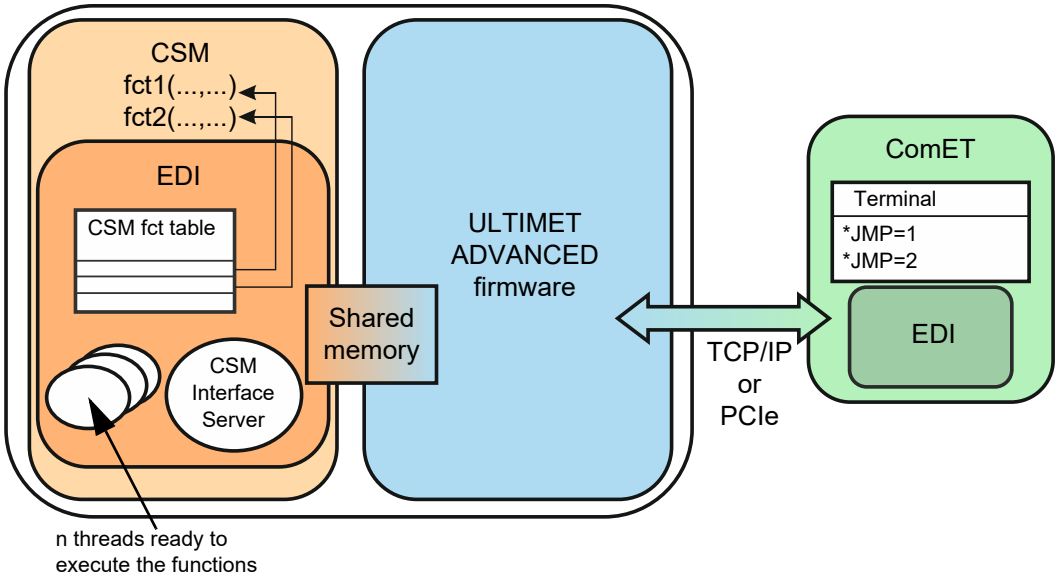
- In the final production environment, the Xilinx SDK is not present, therefore the CSM must start a “Command Interface server” which allows an application (or **ComET**) running on a host PC to execute the high level functions using the *JMP command (refer to §22.3). Of course, if the CSM is running on the emulated Windows environment, this mode is not supported.

The following block diagrams depict these two modes:

- Development environment: executing functions from the Xilinx SDK environment



- Production environment: executing functions from host PC



18.6 Recommendations

- EDI C++ interface cannot be used on FreeRTOS. Therefore, a CSM can only be developed in standard C.
- Since the CSM is based on EDI, all EDI functions can theoretically be used (apart from the few exceptions presented in §24). However, one should not forget that the CSM is an embedded real-time application, and as such, superfluous tasks or tasks not compatible with a real-time execution must be avoided (e.g. a CSM should not be used for configuration management, firmware download or similar functionalities).
- A general recommendation about real-time development is to avoid dynamic memory allocation/ deallocation during the application execution loop. Dynamic memory allocation/deallocation is not only slow, it generates also memory fragmentation. Therefore, the CSM application should run in three phases:
    - Initialization phase: Done only once, at startup of application
      ☺ Dynamic allocation allowed
    - Application execution phase: Generally a loop executing the real job of the application (can run for several months)
      ☹ **Dynamic allocation forbidden**

- End phase: Done only once, just before the application end.
  ☺ Free dynamically allocated resources

Please refer to HTML documentation of EDI and §25 for dynamic allocation in FreeRTOS. The functions doing dynamic allocation are indicated with: **Warning: Allocate Heap memory**

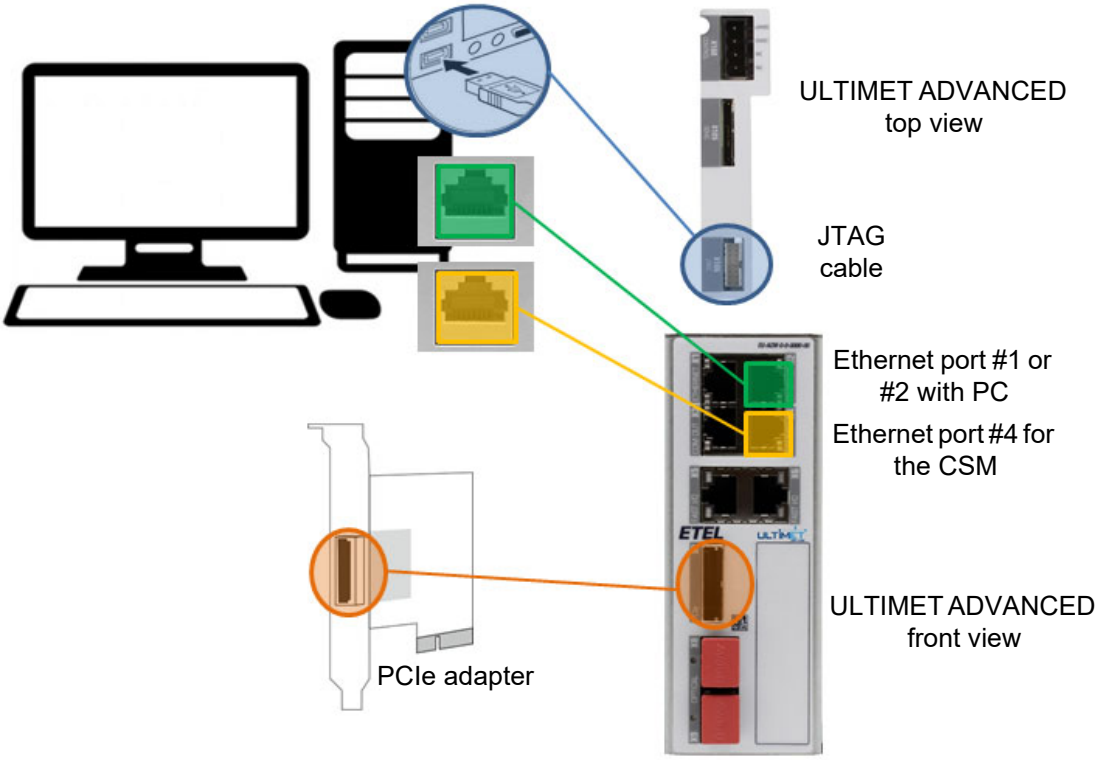
- Another recommendation is to avoid using waiting function inside a real-time handler. EDI allows to attach a user handler which can be called synchronously with the TransnET cycle (100 µs). Therefore, the user should avoid to call logging functions (dsa_csm_printf) or waiting functions (dsa_wait…) inside its handler. Even more, only the “release” version of the application will be able to follow this high frequency.

19 Development environment

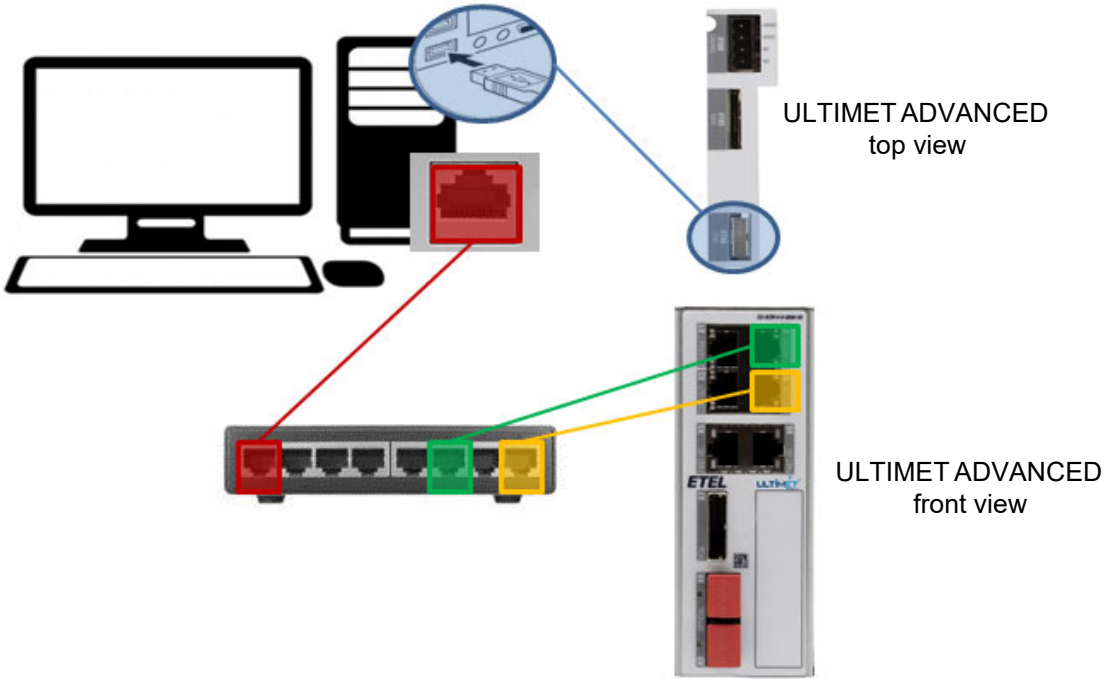
19.1 Overview

To develop a Customer Software Module for ULTIMET ADVANCED, you must have the following configuration:

- The Xilinx SDK (XSDK) installed on a Windows PC (Windows 10).
- The ULTIMET ADVANCED installed in a rack and powered by an adequate power supply (refer to the corresponding “Hardware Manual”).
- The ULTIMET ADVANCED connected to the development PC with the JTAG cable. It is used by the Xilinx SDK to download, run and debug the CSM. The JTAG connection is not needed in production environment. In development phase, JTAG is also used as standard input-output for all stdio C functions (printf/scanf). Please refer to §23.1 for more information about this connection and its limitations.
- Optional: The ULTIMET ADVANCED CSM connected to the PC with the Ethernet port#4. This Ethernet port is reserved for a direct connection from CSM to PC, and principally used to replace the slow JTAG standard input-output interface. All logs generated by the CSM using EDI dsa_csm_debug_printf will be redirected to this Ethernet port. On the PC side, you can connect to this port using a Telnet session to see the generated logs. Moreover, Telnet input can be requested using EDI dsa_csm_debug_scanf function.
- Optional: The ULTIMET ADVANCED firmware connected to the PC application (ComET) using:
    - Ethernet port#1 or port#2. Ethernet ports reserved for a direct connection from PC to ULTIMET ADVANCED firmware.
    - PCIe: Used for a direct connection from PC to ULTIMET ADVANCED firmware. This solution is more time deterministic than the Ethernet ports.



Of course, the 2 ULTIMET ADVANCED Ethernet ports can also be connected to a single Ethernet port of the PC using a switch:



As mentioned in §18.2, the CSM can be built for Windows to help testing and debugging. To do so you need to get and install Microsoft Visual Studio 2019. Please refer to the Microsoft documentation for how to get and install this environment.

19.2 CSM template package

19.2.1 Content

The CSM template is provided as a .zip file (CSMTemplate.zip) containing:

- The projects for two development environments:
  - FreeRTOS for the Xilinx SDK environment. For this environment, you can find:
    - The project definitions
    - The hardware support package
    - The EDI libraries for FreeRTOS organised in two groups:
      - The Production FreeRTOS Software Platform
      - The Development FreeRTOS Software Platform

    More details can be found in §19.3.2.2.

  - VStudio20XX for Windows Visual Studio 20XX. For this development environment, you will find:
    - the Visual Studio solution and project all set up to be compiled under Windows,
    - The EDI libraries for Windows.

- The source code directory (common to Windows and FreeRTOS).

19.2.2 Installation

To make the CSM template available, unzip the CSMTemplate.zip file on your hard disk.

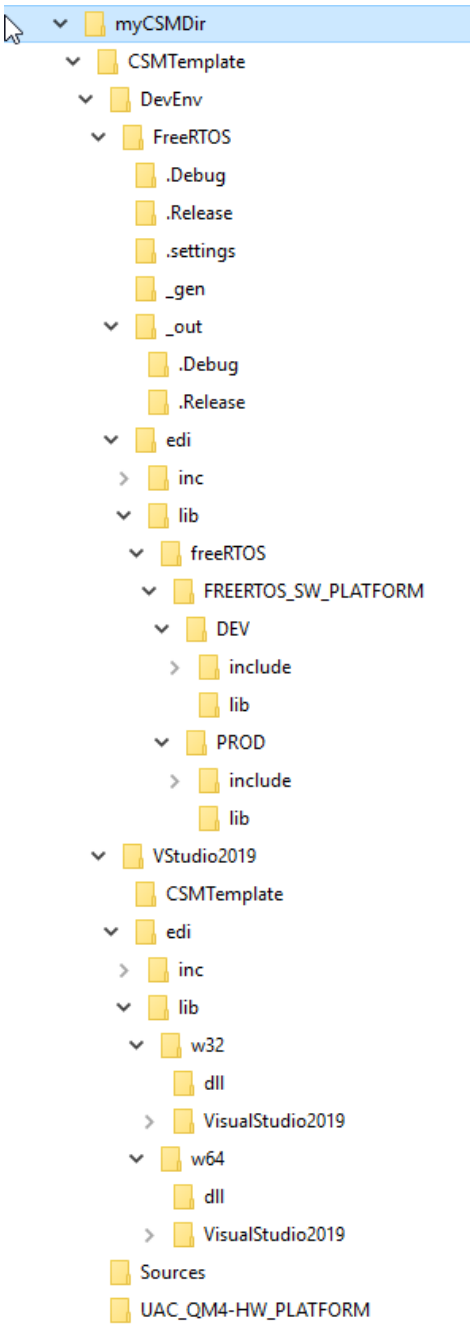
In the rest of this document, it is assumed the package was unzipped directly in C:\ and the template project directory in the zip file is named myCSMDir so you might see C:\myCSMDir in some screenshots.

If it is unzipped elsewhere, just replace in the following sections C:\ by wherever it was unzipped. Also, you can change:

- The name of the myCSMDir directory to reflect your application, but do that just after unzipping the file and before proceeding with the setup.
- You can also change the name of the name CSMTemplate for the project but do that from within the development environment so that all necessary references are updated.

- Refer to §19.3.2.1.2 to see how to do that in the SDK environment

The extraction will create the following folder structure:

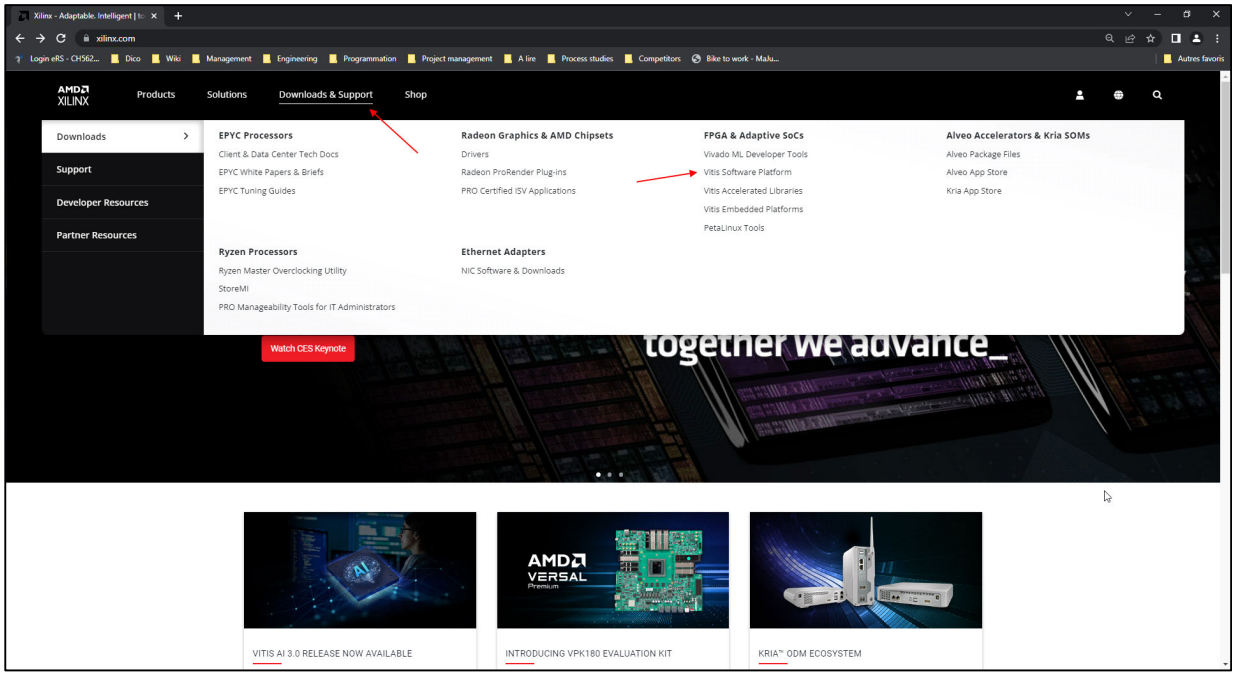


19.3 Working with the Xilinx SDK

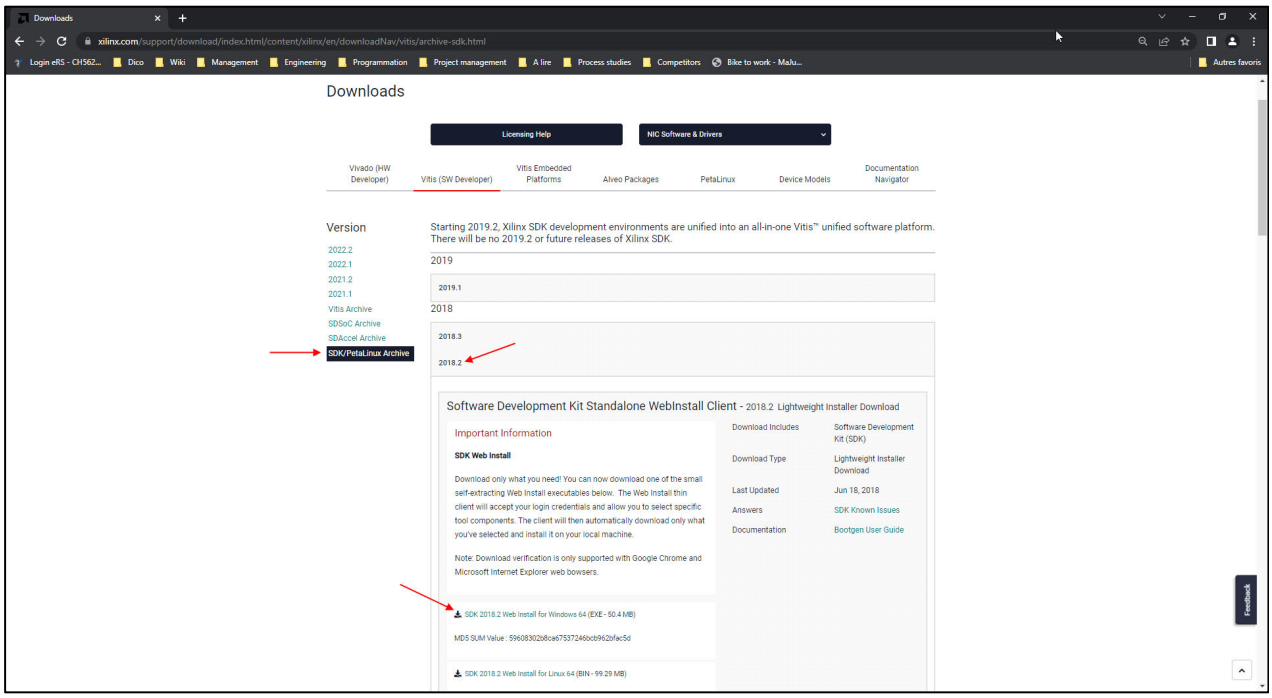
19.3.1 Installation

Remark: To avoid installation issues, deactivate any UltimET driver prior to installing the Xilinx SDK.

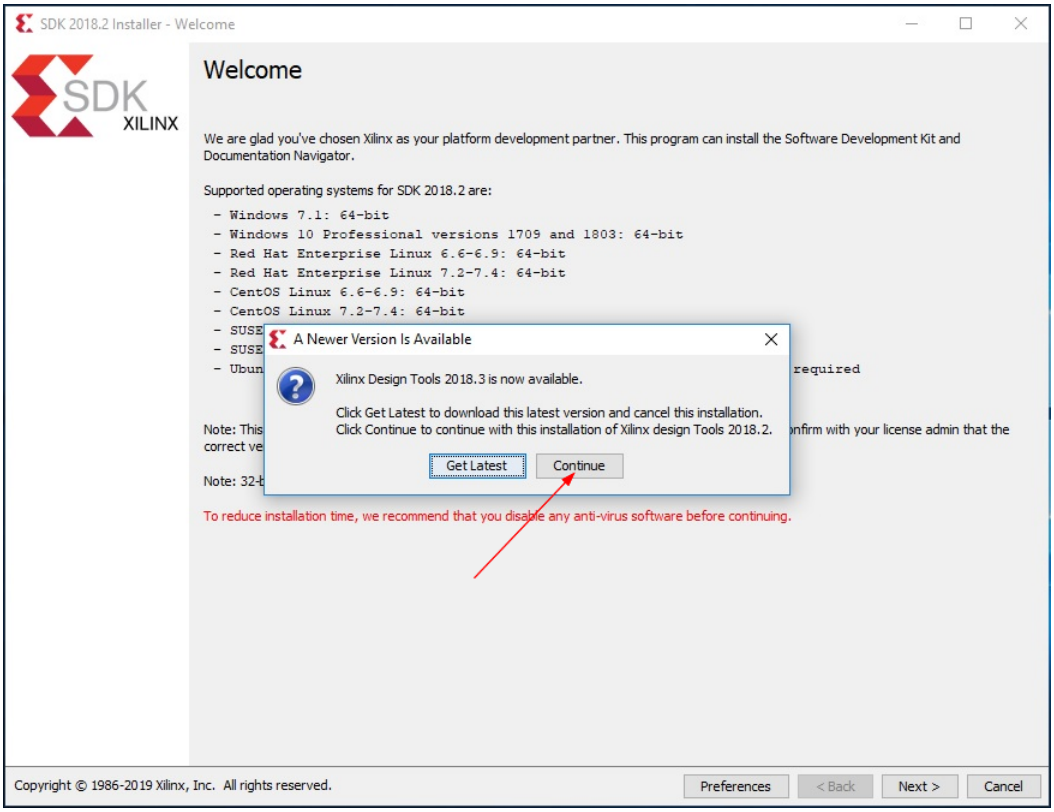
The Xilinx SDK installation can be found on the Xilinx Web Site: www.xilinx.com. It is a free software (no license is required), however a user account must be created in order to download it. Once the user account has been created, go to the “Download & Support” tab and choose the “Vitis Software Platform” option:



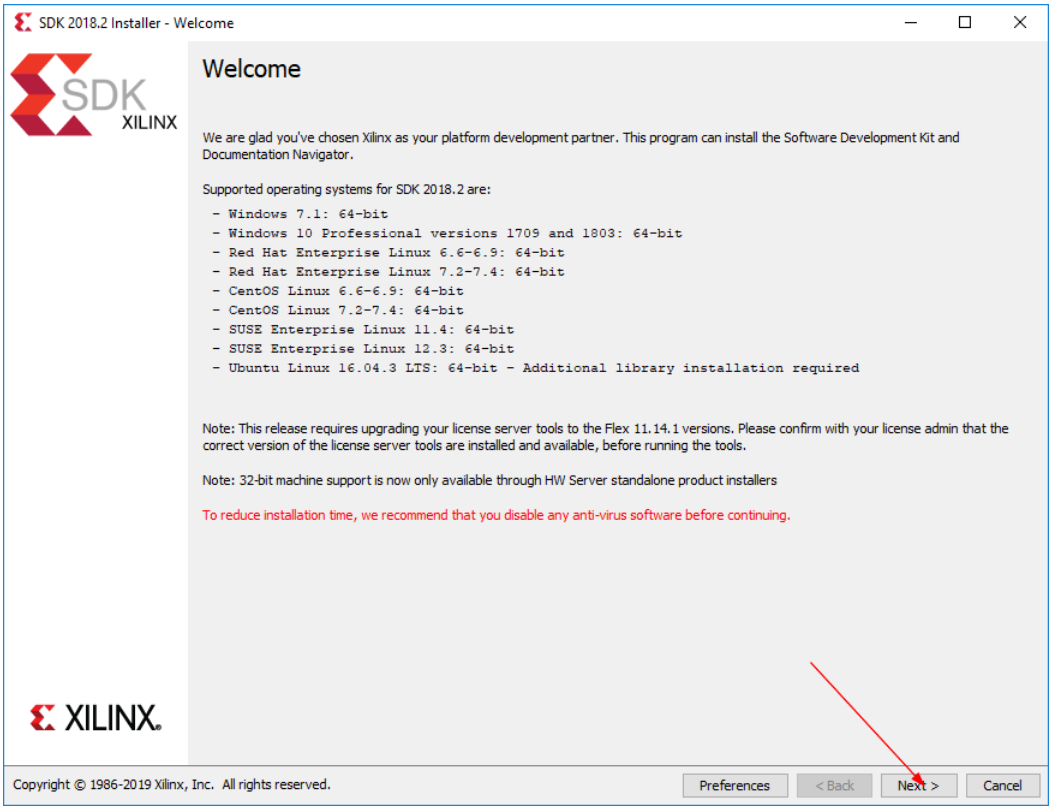
On the “Downloads” page, check the “Vitis (SW Developer)” tab is selected, then under “Version” click on “SDK/ PetaLinux Archive” and choose the 2018.2 SDK version for downloading:



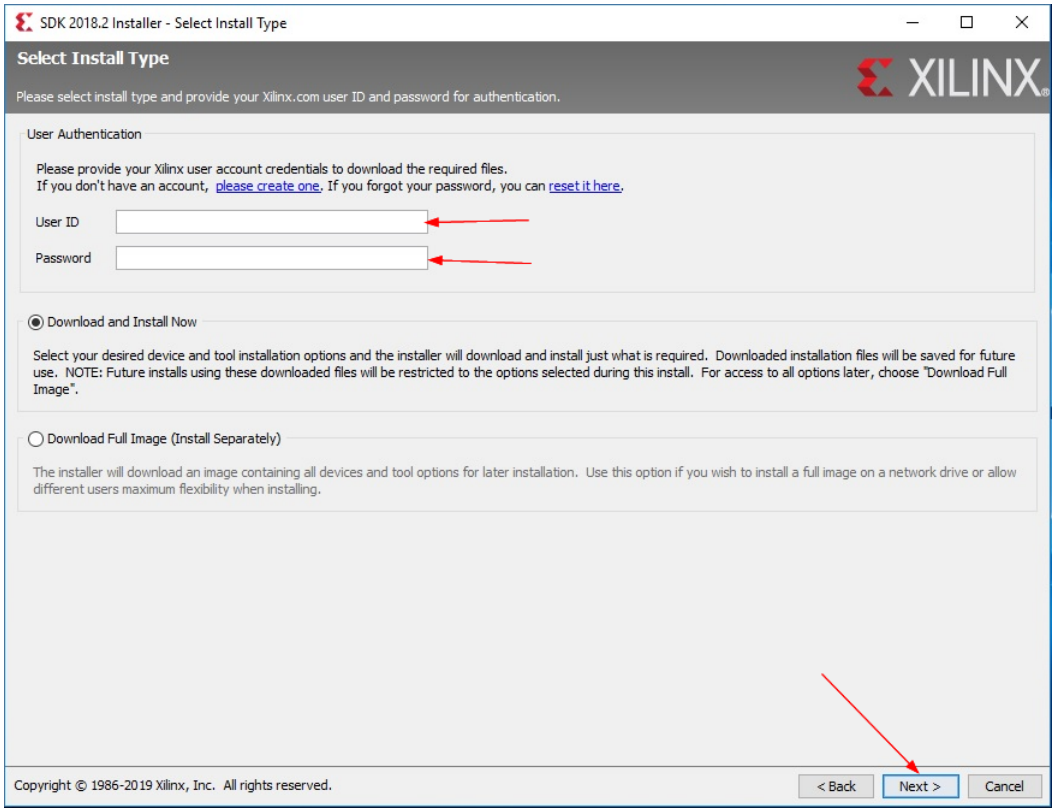
Run the installation process. The process will start by informing that a newer version is available. Proceed with the previously selected version by choosing “Continue”:



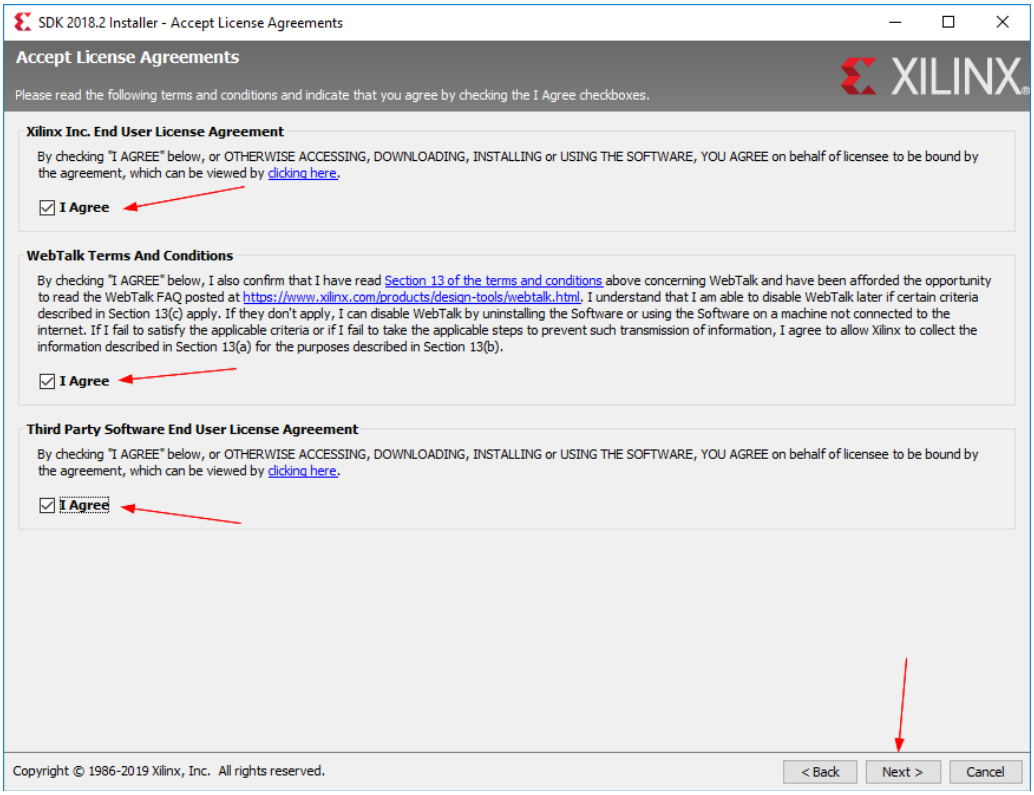
Then press “Next” to begin the installation:



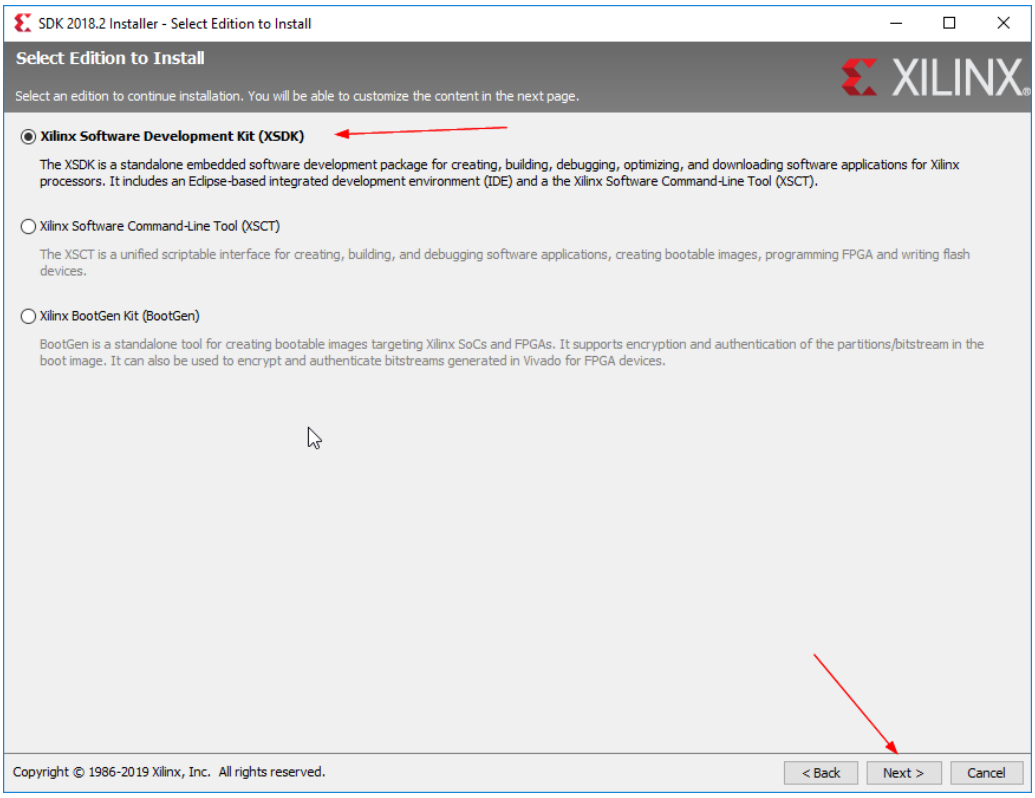
Enter your user ID and password:



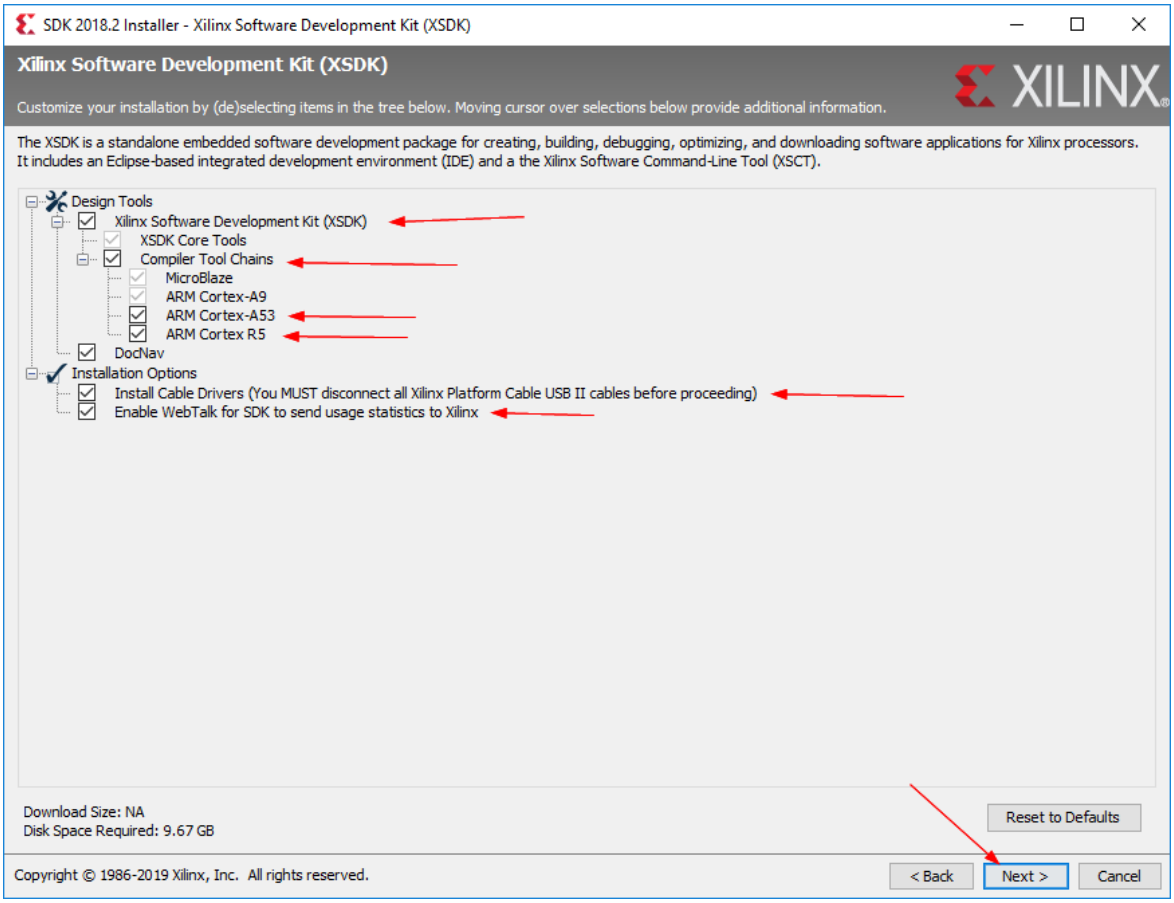
Read and accept all license terms. Press “Next” to proceed.



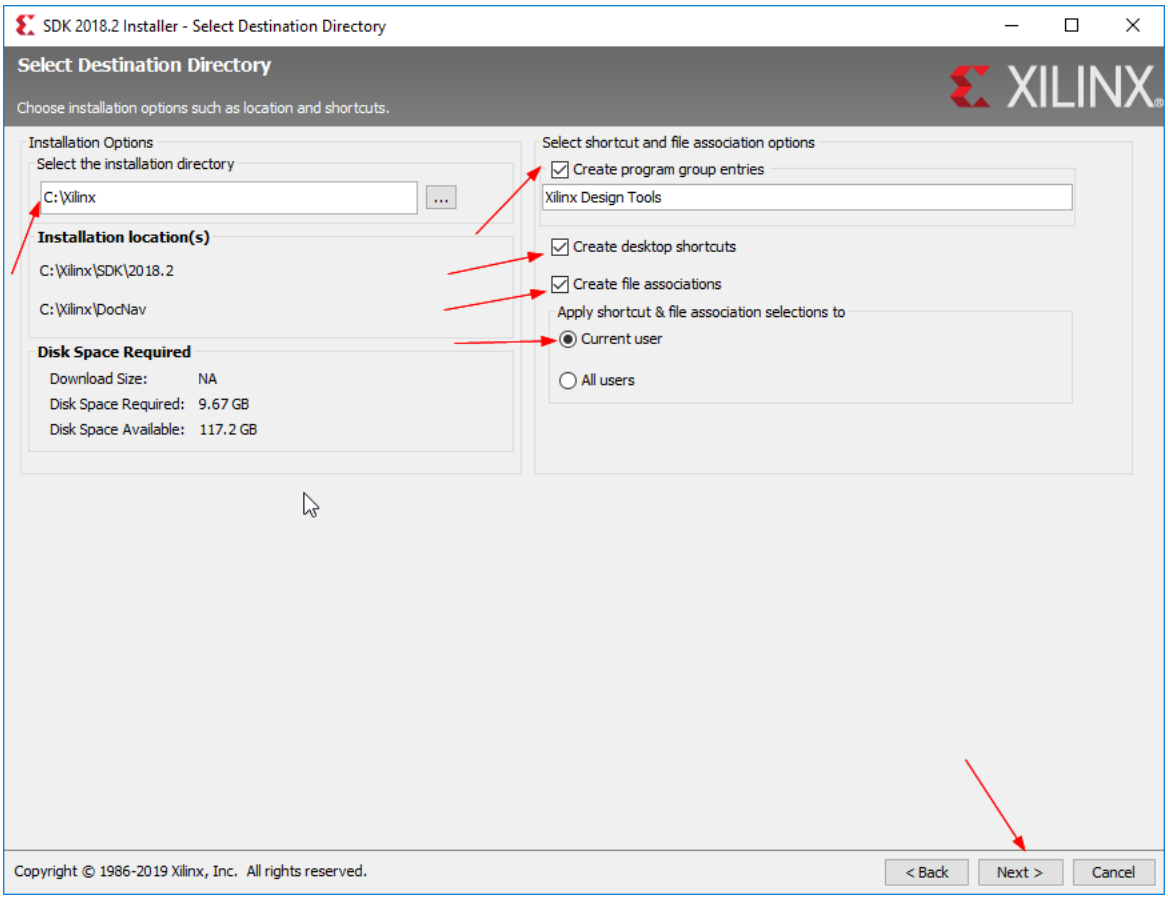
Then select the first option “Xilinx Software Development Kit (XSDK)” and press “Next”:



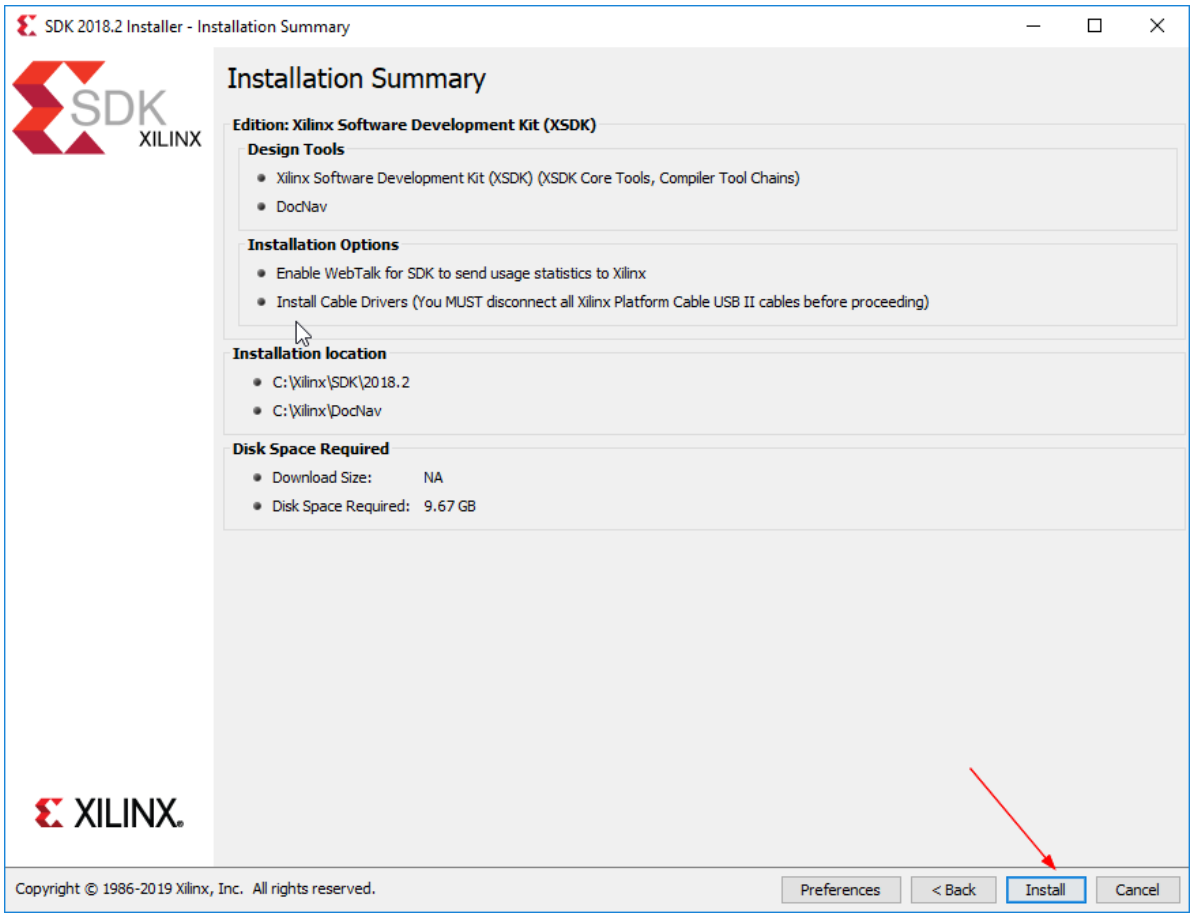
Do not modify the default options. Be sure that the JTAG cable is NOT connected and press “Next”:



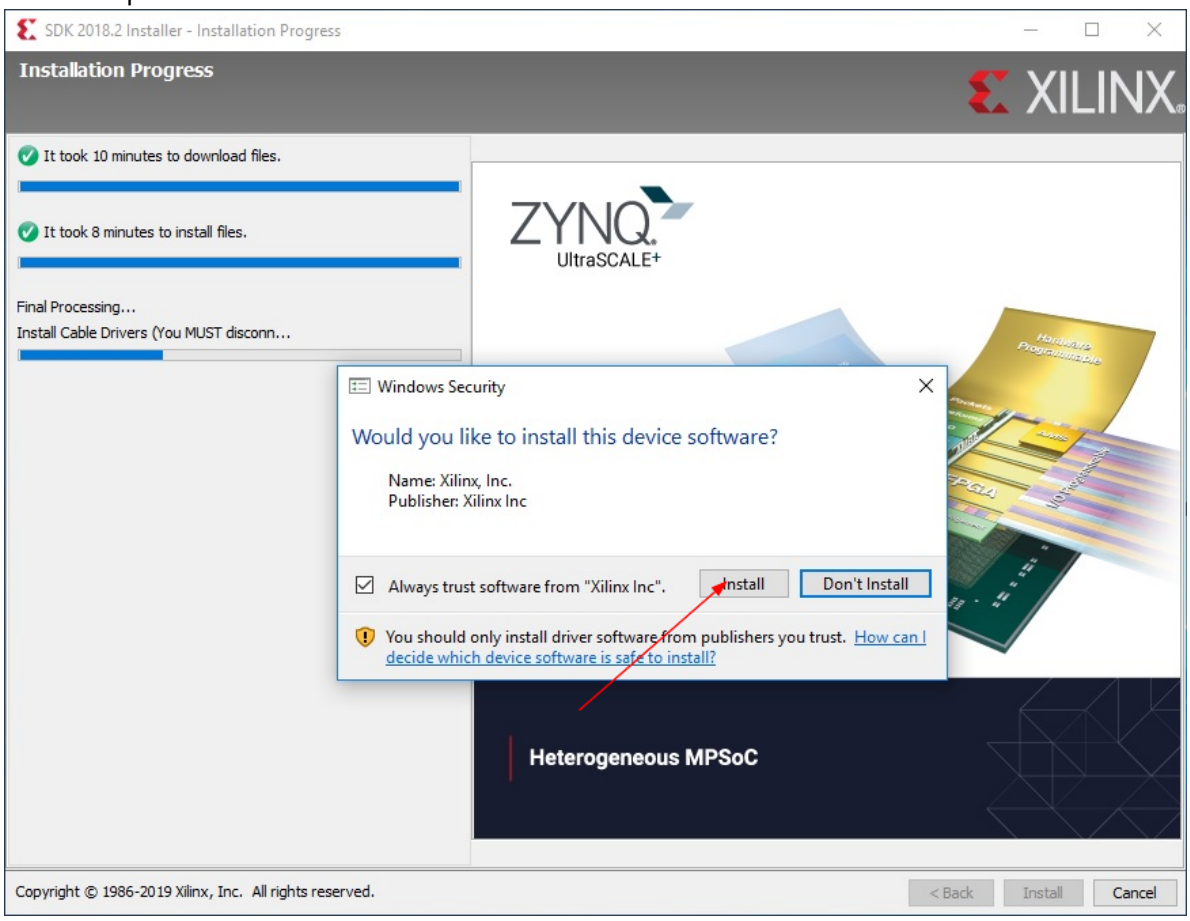
Choose the destination directory and press “Next”:



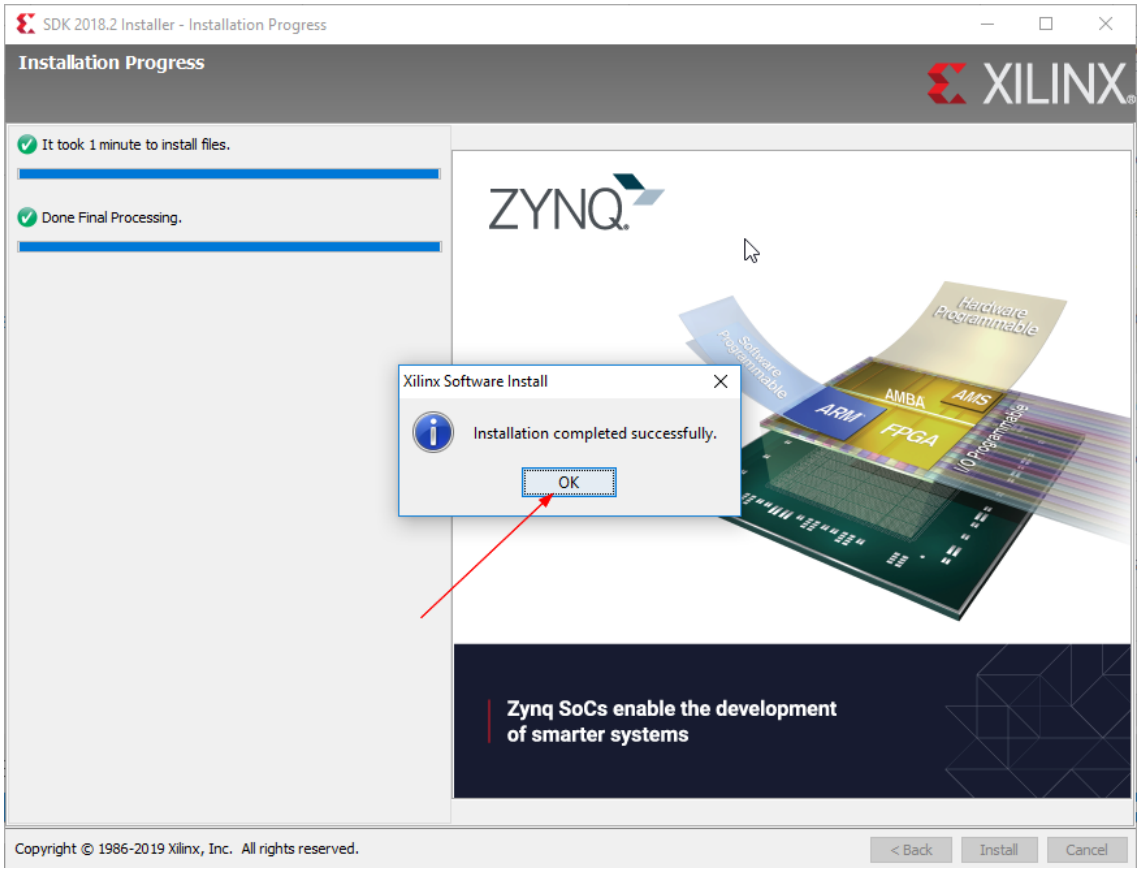
Check the installation summary and choose “Install”:



Finally, the installation process asks if you want to install the JTAG driver. If not done yet, disconnect the JTAG cable and accept the driver installation:



Once the installation is finished, you get the successful message depicted below. Just press “OK” to conclude the process.



19.3.1.1 Patching

Once the installation completed, the original Xilinx SDK must be upgraded with an ETEL specific patch. This patch is provided as a .zip file (SDKXilinxPatch2018.2.zip) and provides specific tools for the development of ULTIMET ADVANCED CSM. This .zip file can be found directly in the directory where you have unzipped the CSM template .zip file (refer to §20).

To patch the SDK:
1. Go in the directory where you have installed the SDK. By default, the directory is C:\Xilinx.
2. Copy the SDKXilinxPatch2018.2.zip in this directory.
3. Extract the .zip file in the directory

19.3.2 Setting up

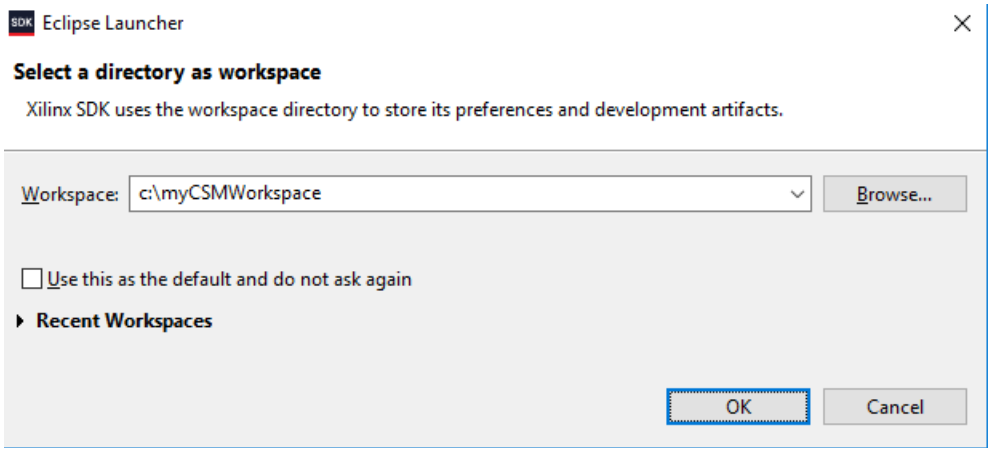
19.3.2.1 SDK Workspace

19.3.2.1.1 Creating the workspace

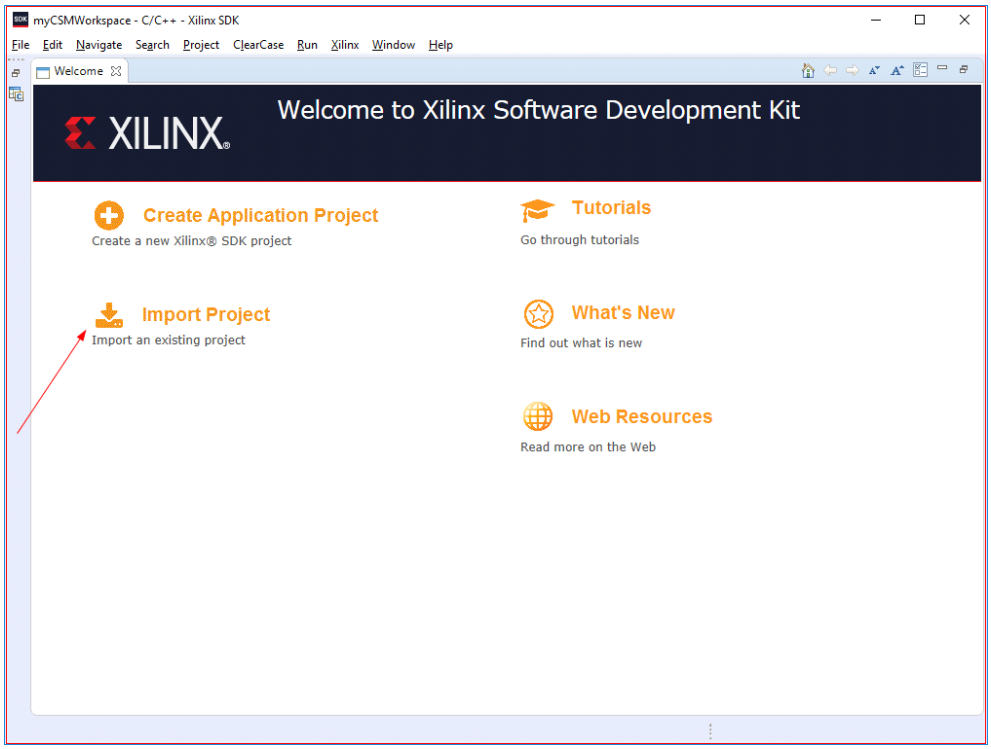
To work in the SDK development environment, you need to create a “workspace” that will hold the CSM template project. The workspace and the project location are unrelated. A workspace can contain several projects.

To create a workspace:
1. Launch the Xilinx SDK environment.

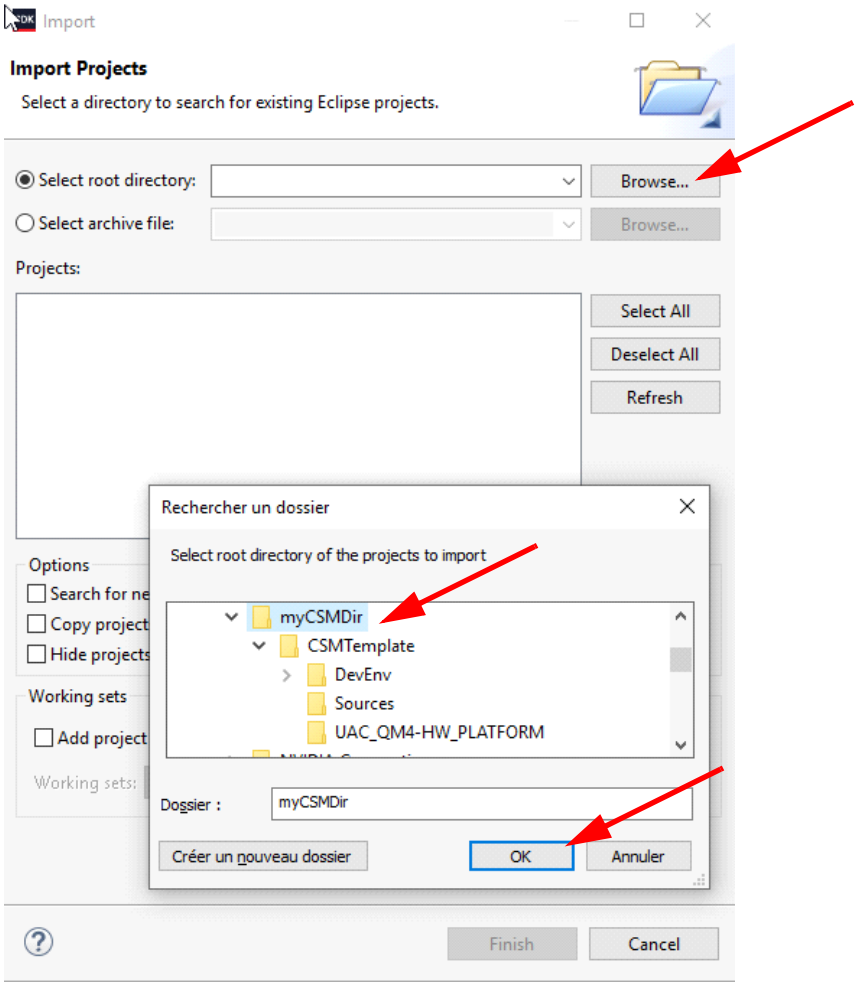
2. Choose the directory in which you want to store your workspace. It does not have to be related to where you unzipped the CSM template zip file.



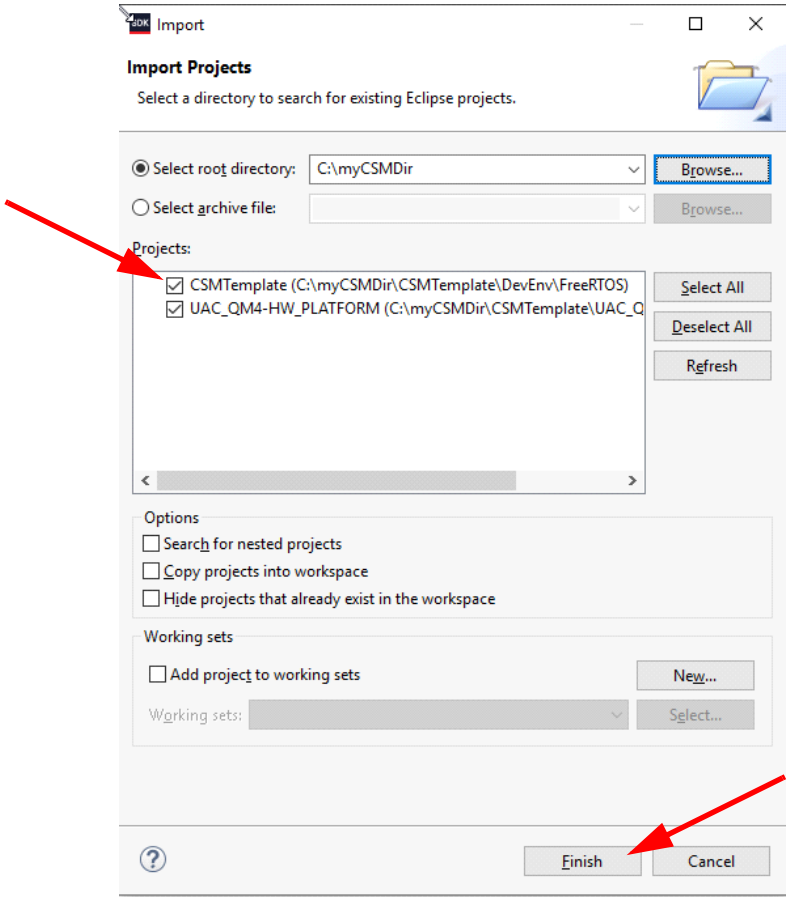
3. Import the CSMTemplate project:



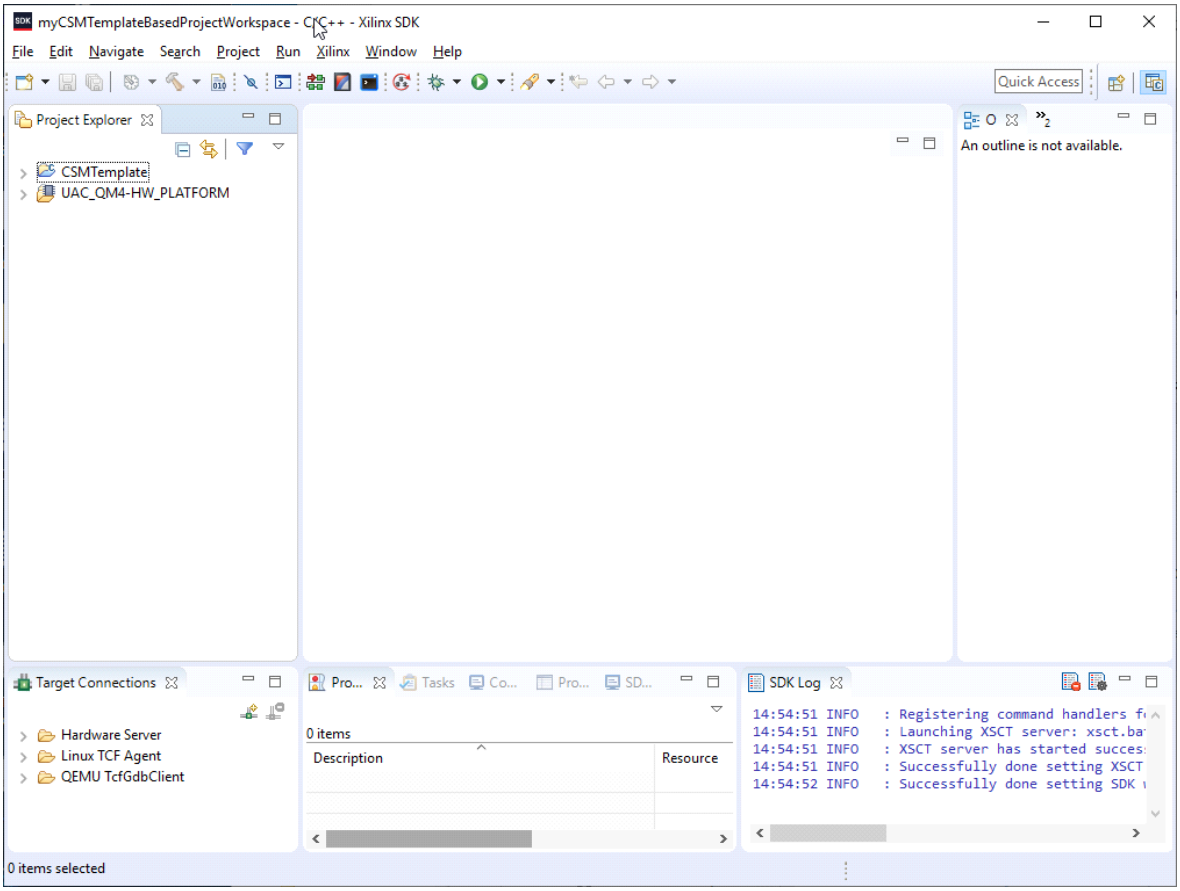
4. Browse the directory where you have extracted your CSMTemplate (the Xilinx SDK will display the projects that can be imported):



5. Select all projects and press “Finish” for importing:



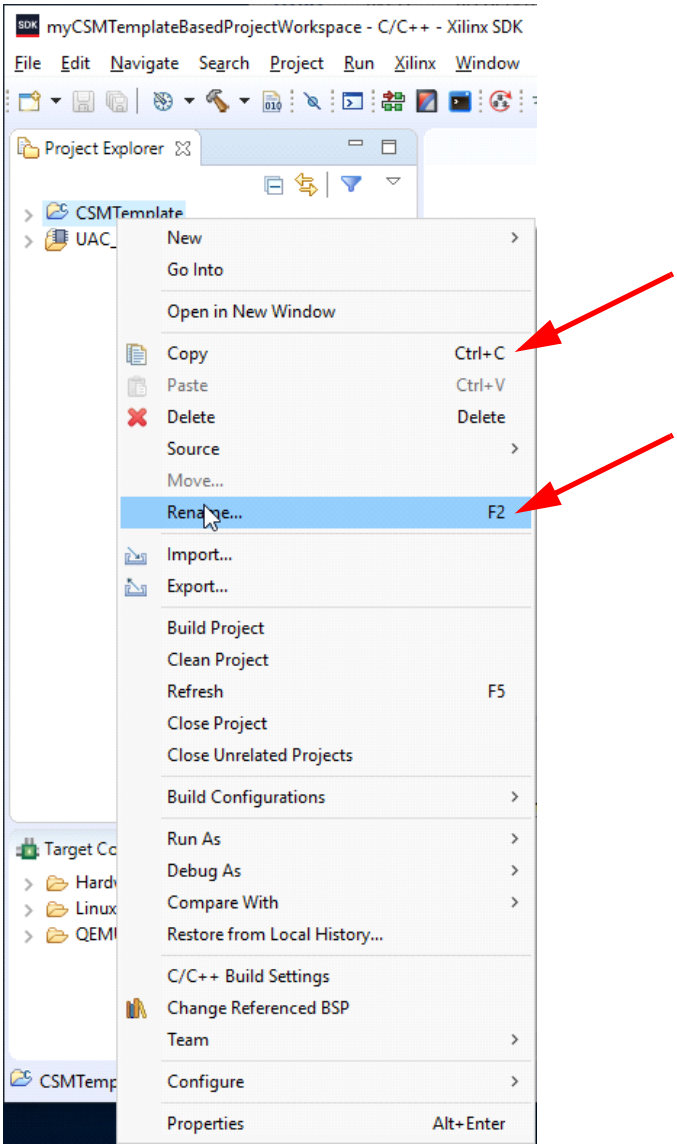
6. Your environment should now look like the screenshot below. If not, restart the process and check again all dependencies:



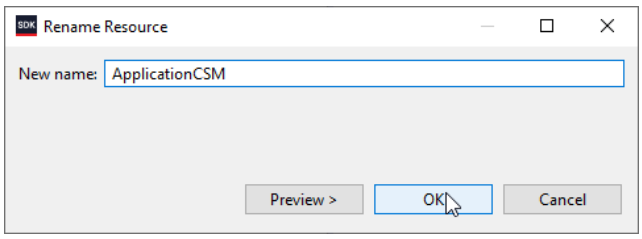
19.3.2.1.2Renaming the CSMTemplate project

If you want to rename or copy the CSMTemplate project:

1. Right click on “CSMTemplate” in the “Project Explorer” tab of the SDK, and select “Rename” or “Copy”:



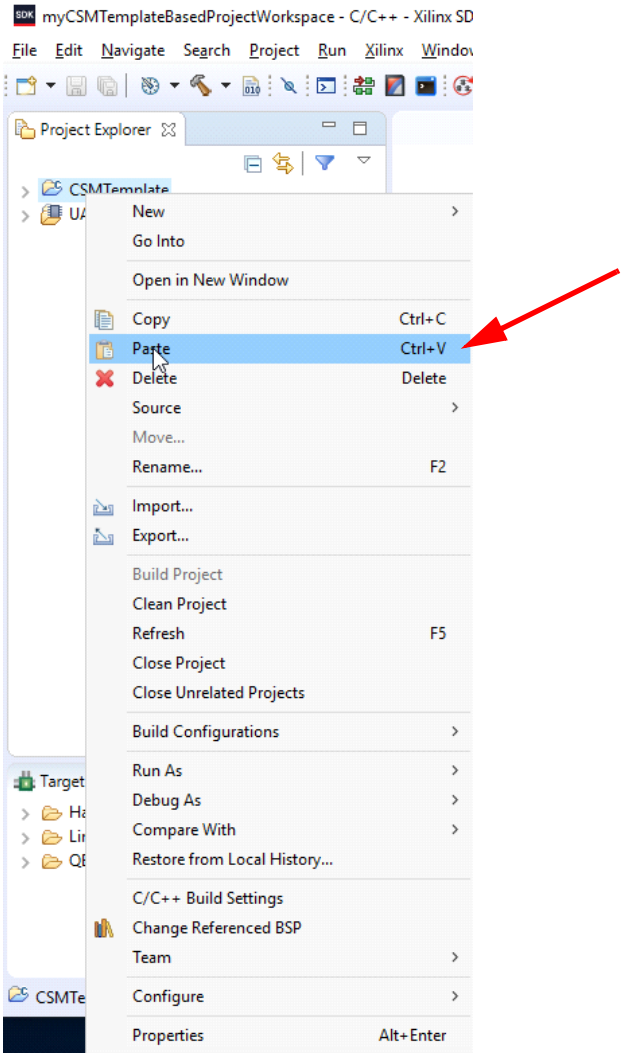
If renaming, type in new name:



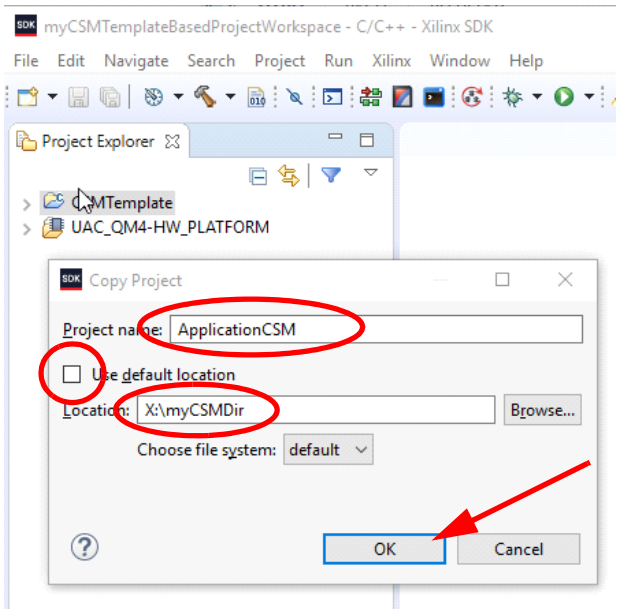
Warning: Xilinx SDK issue and workaround: renaming of the project modifies the relative path of sources. A workaround has to be applied:

- Open project properties (right click on the project in Project Explorer).
- Open the Resource level and click on "Linked Resources".
- Click on "copy_Parent" and then on the edit button.
- Replace the text in the location field with "${PROJECT_LOC}\..\..".
- Click OK twice.
- Your project is now up-to-date and can be built.

2. If copying, right click again on “CSMTemplate” in the “Project Explorer” tab of the SDK, and select paste:



3. Enter new name and location:



4. Type in the new name, and a location for the new project.
   The default location the SDK chooses is in the workspace. You can deselect “Use default location” and place the new project next to the template provided, in “myCSMDir” as above, or anywhere else to your liking.

19.3.2.2 CSM template SDK project

For the Xilinx SDK FreeRTOS environment, the CSM template contains:

- The Hardware Platform definition UAC_QM4-HW_PLATFORM, required for debugging.
- Two FreeRTOS Software Platform which contains the FreeRTOS and Xilinx libraries and the include files required to compile a CSM:
    - The Production FreeRTOS Software Platform FREERTOS_SW_PLATFORM\PROD is used when a CSM is linked for the Production Mode (REMOTE_CONTROLLED is defined). This software platform package does not have standard Input/Output. All stdio C functions will be without any effect (printf, scanf, …).
    - The Development FreeRTOS Software Platform FREERTOS_SW_PLATFORM\DEV is used when a CSM is linked for the Development Mode (REMOTE_CONTROLLED is not defined. Refer to §19.3.2.3 to check your defined pre-compilation symbol). The standard Input/Output of this package is the JTAG. All stdio C functions use the JTAG console.

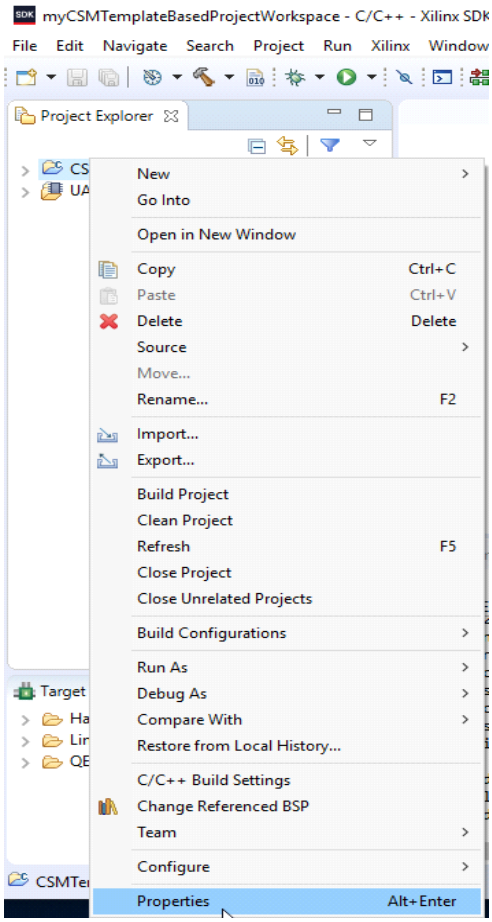
      Therefore, it is mandatory to have the JTAG connection opened when using this Software platform.

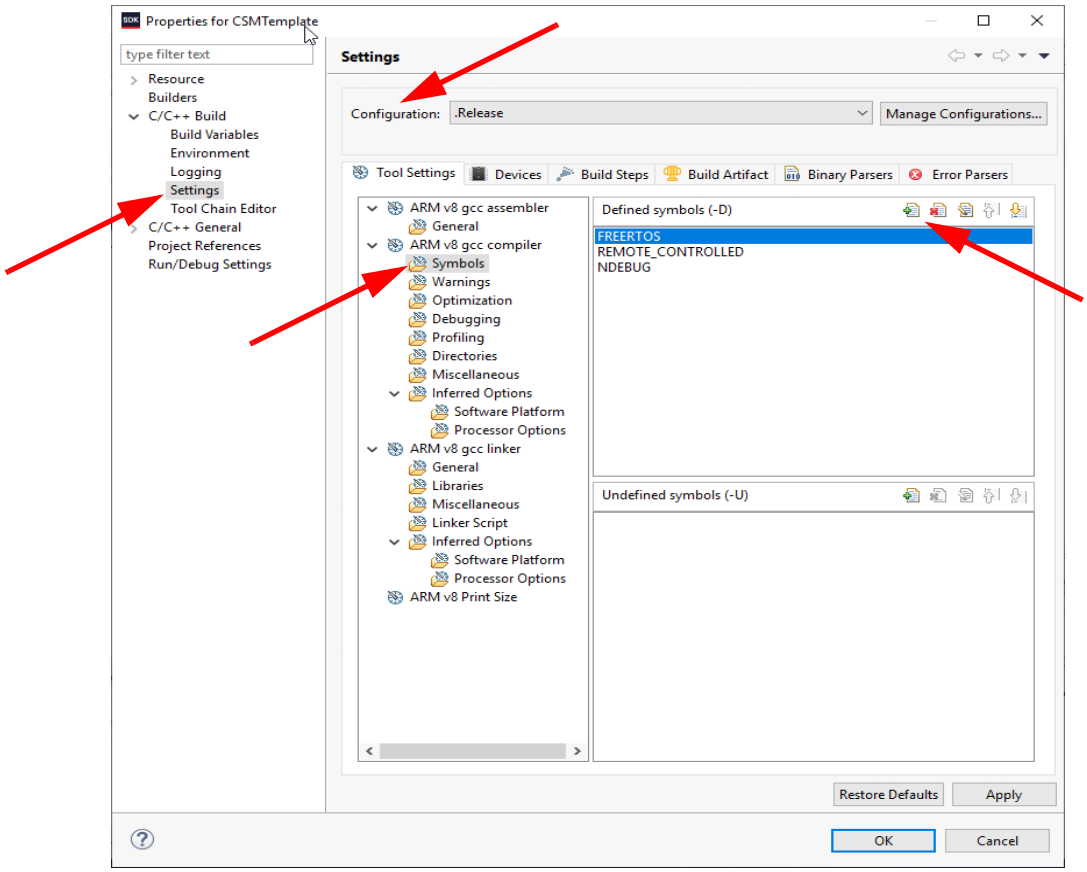
By default, the CSMTemplate “release” version is configured to use Production FreeRTOS Software platform and the “debug” version configured to use Development FreeRTOS Software platform.

19.3.2.3 Modifying environment variables and symbols

As mentioned in §19.1, all the project definitions are relative to the project directory, so no matter where you unzipped the CSM Template package, you do not need to modify any settings to compile the project.

You might however want to add compiler symbols specific to your CSM application. To do so, right click on the project in the “Project Explorer” tab of the SDK and select “Properties”.





You can add your own symbols by clicking on the little button.

The predefined symbols shown when opening this window are set as necessary in the CSM Template package:

- The symbol **FREERTOS** is mandatory. It is used to compile the FREERTOS version of all EDI header files.
- The symbol **REMOTE_CONTROLLED** is defined when compiling the release version used for production mode. The CSM template will start the necessary components allowing to be remotely controlled via JMP commands. In debug mode, the symbol **REMOTE_CONTROLLED** must NOT be defined (you will see it is suffixed with an underscore, in order to be easily reused).
- The symbol **NDEBUG** is defined when compiling the release version. It avoids adding some pieces of code (e.g. logging) used in development mode only.

19.3.3 CSM version and description information

A version, name and description for the CSM is included in the generated output file. Once downloaded onto the ULTIMET ADVANCED, this information can be queried by reading the following monitoring registers:

- *M282:7 returns the version of the CSM;
- *M284 returns the name of the CSM;
- *M285 returns the description of the CSM.

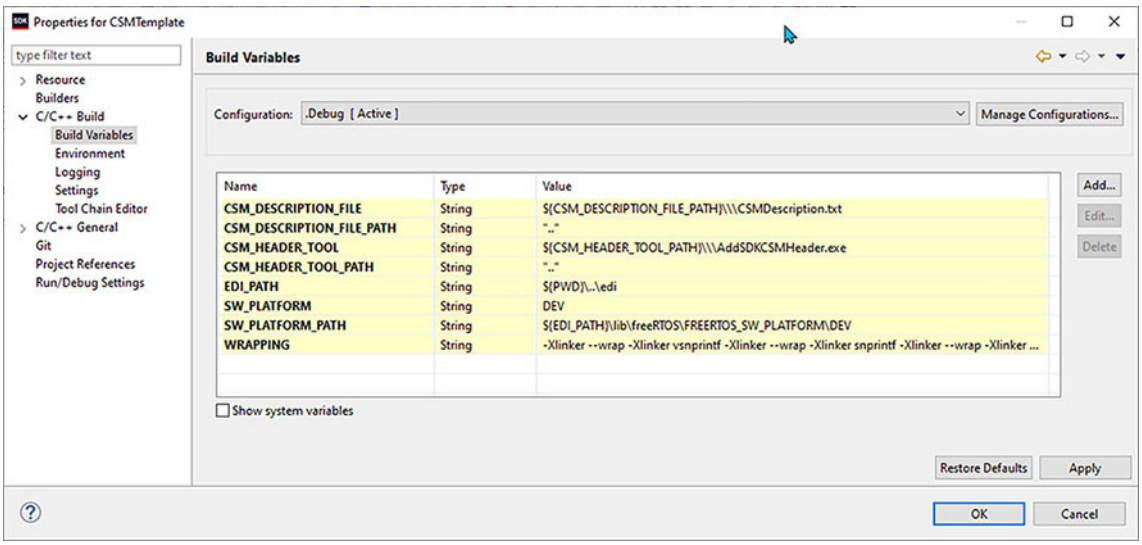
The information is read from a text description file called CSMDescription.txt that you can find in ...myCSMDir\CSMTemplate\DevEnv\FreeRTOS.

You can edit this file to update it with the relevant information. Beware that "Version: ", "CSM Name: " and "CSM Description: " (including trailing whitespaces) are keywords and must NOT be changed.

In addition, the software platform as defined in the CSM project settings and the debug/release compilation options are also injected in the header of the generated output file. This information can be read in monitoring registers *M286 for the platform used and *M287 for the compilation options. The header containing this information is injected in the output file when building the project.

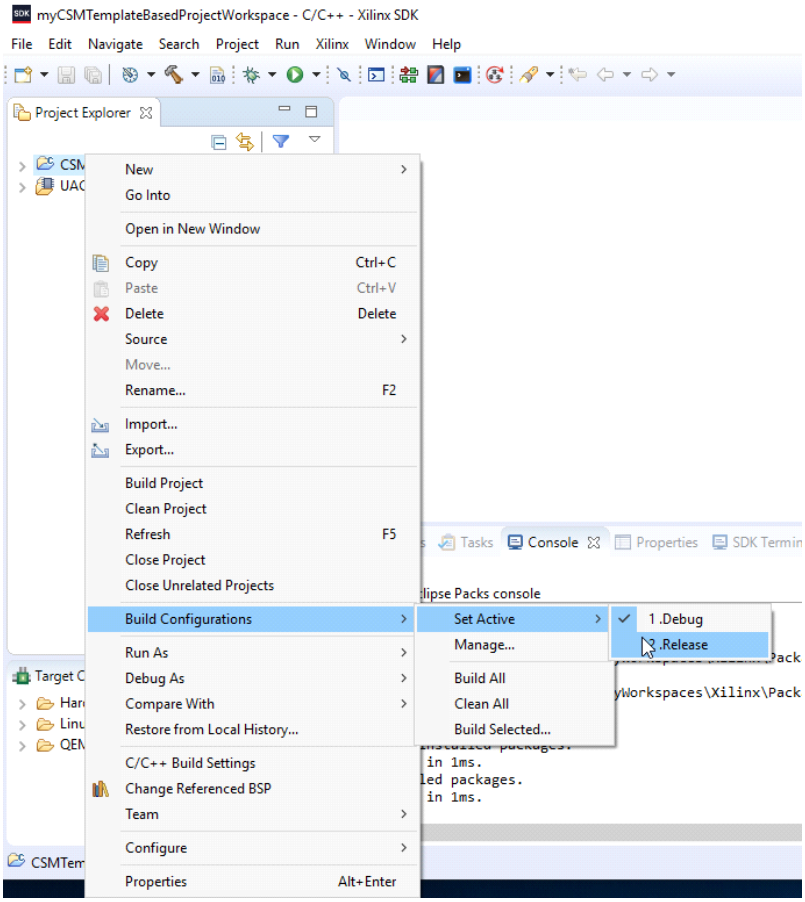
Remark: You can change the location and name of this file if needed by updating the project's settings build variables SW_DESCRIPTION_FILE_PATH (make sure it is enclosed in quotes) and/or SW_DESCRIPTION_FILE.

The tool that is called to add the header is also ...myCSMDir\CSMTemplate\DevEnv\FreeRTOS. If this location is unsuitable, you can also change it by updating the project's settings build variable CSM_HEADER_TOOL_PATH (make sure it is also enclosed in quotes).



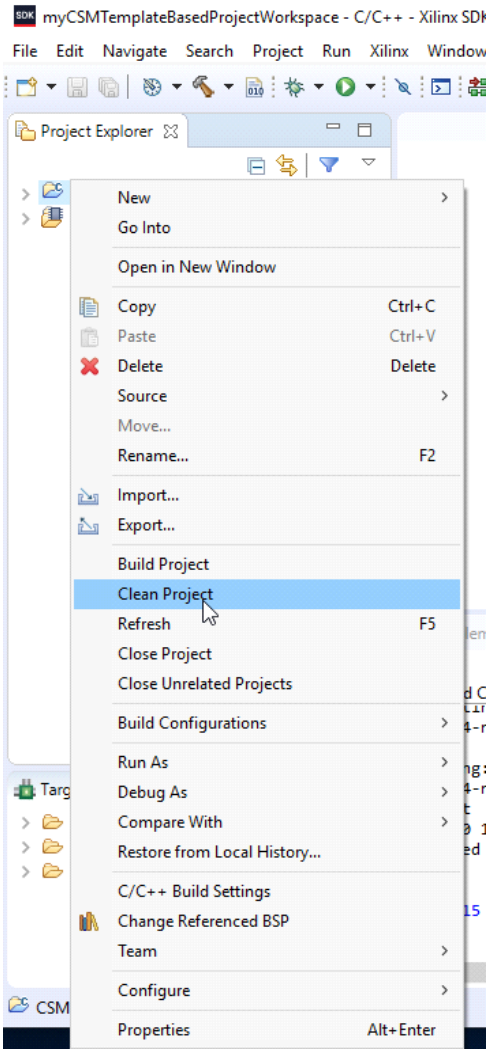
19.3.4 Compiling and linking

1. Choose the version (Release or Debug) to compile by right clicking on the CSMTemplate project and choosing Build Configuration → Set Active → Release or Debug:

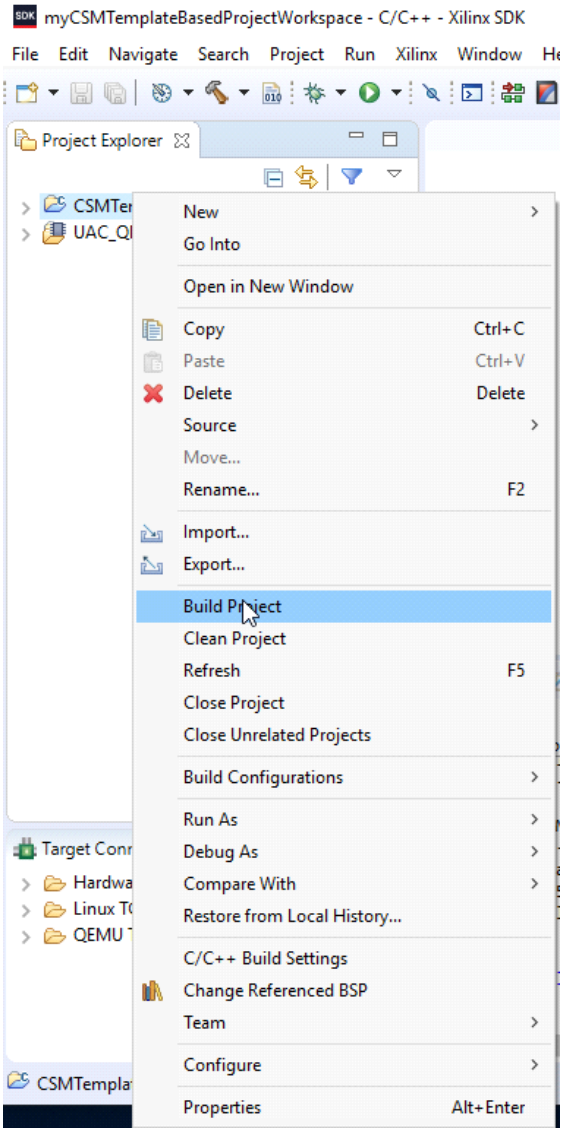


2. Clean the selected project version before compiling by right clicking on the CSMTemplate project and

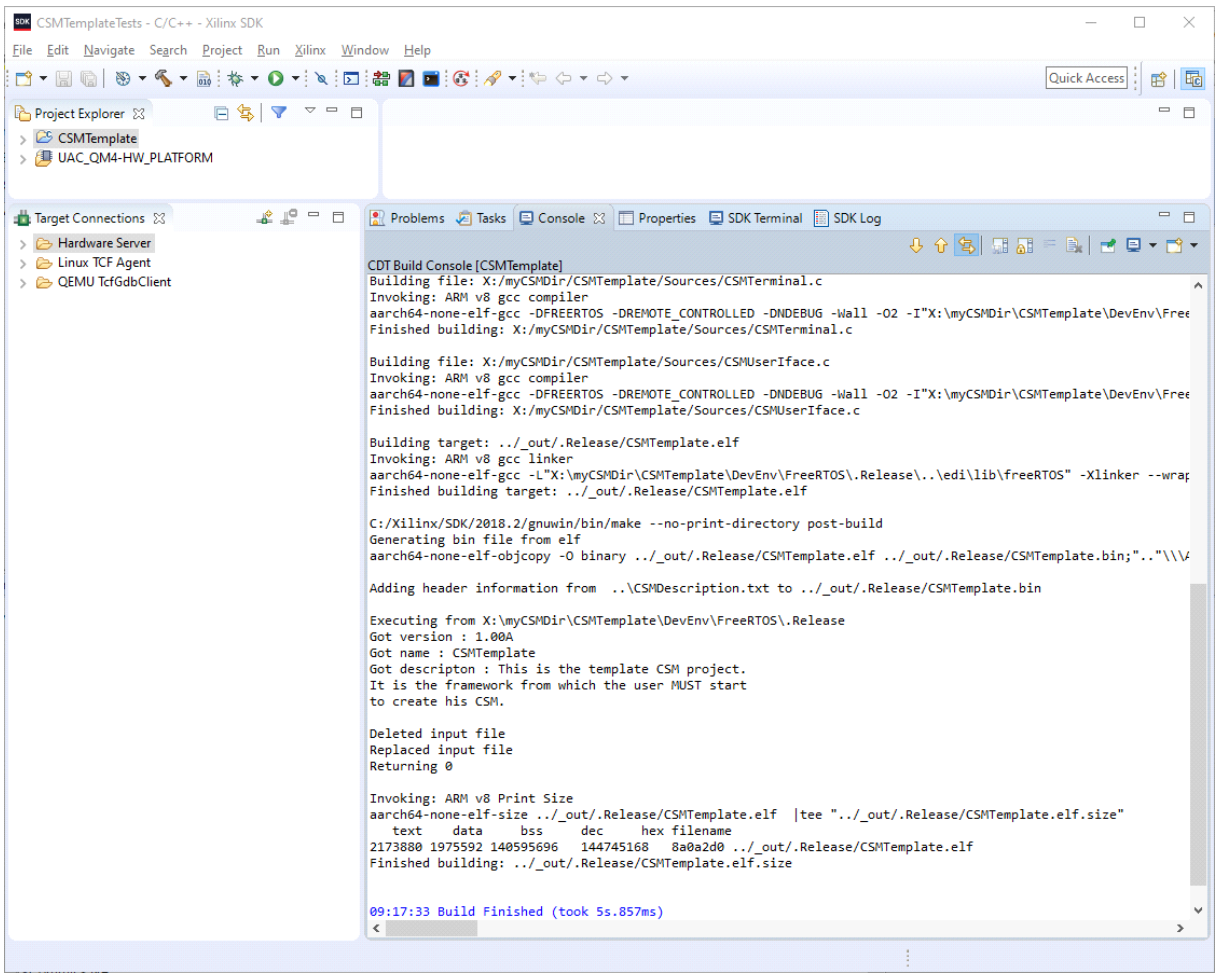
choosing the option “Clean Project” (this will only clean the selected project version: Release or Debug). This operation should be done each time you change the project setting.



3. Compile and link the project by right clicking on the CSMTemplate project and choosing the option “Build Project”:


myCSMTemplateBasedProjectWorkspace - C/C++ - Xilinx SDK
File Edit Navigate Search Project Run Xilinx Window
Project Explorer
New
Go Into
Open in New Window
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Source
Move...
Rename... F2
Import...
Export...
Build Project
Clean Project
Refresh F5
Close Project
Close Unrelated Projects
Build Configurations
Run As
Debug As
Compare With
Restore from Local History...
C/C++ Build Settings
Change Referenced BSP
Team
Configure
Properties Alt+Enter

4. Finally, the compilation output results can be seen on the console:



19.3.5 CSM template execution

The CSM can be executed in two ways:

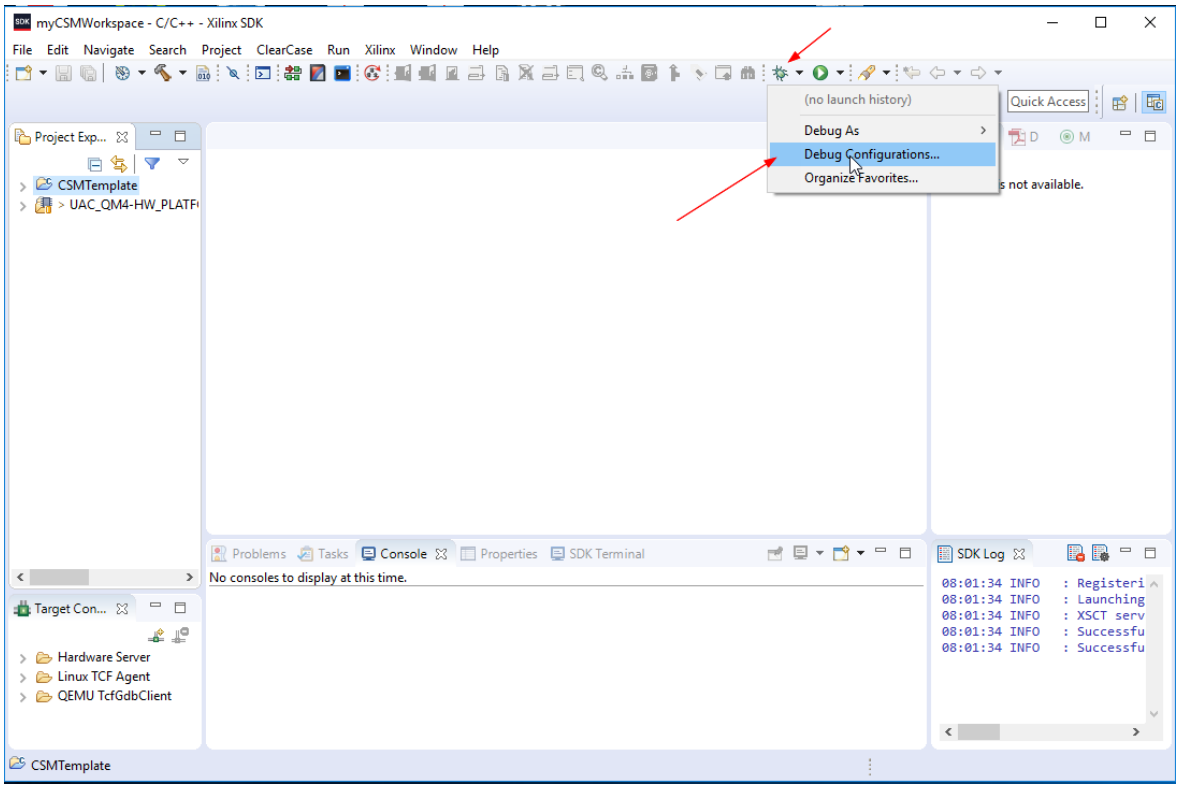
- From the Xilinx SDK environment
- From a host application (i.e. machine software or ComET software)

19.3.5.1 Executing from the Xilinx SDK

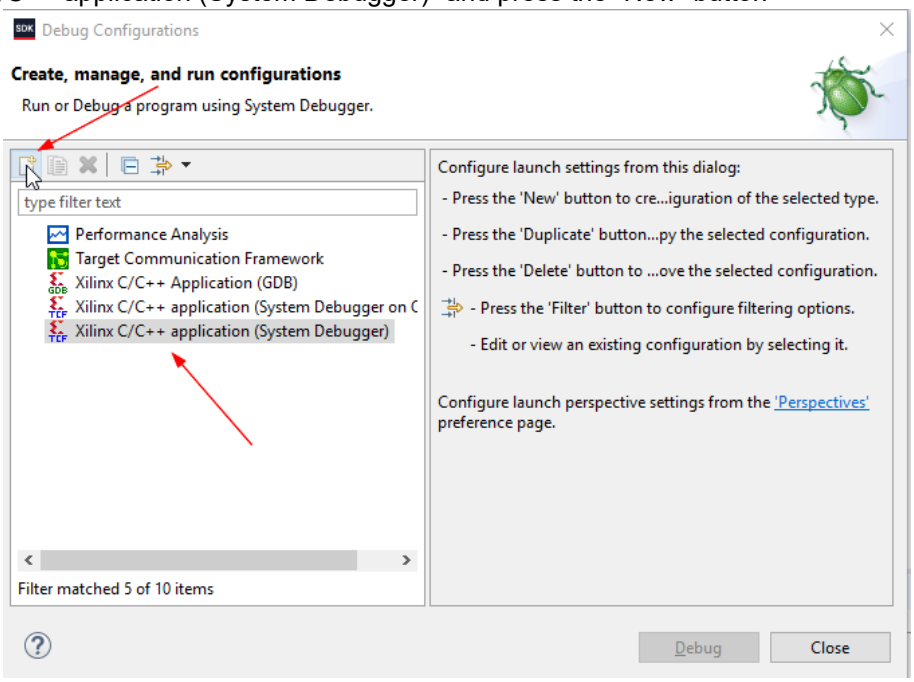
To run a CSM from the Xilinx SDK environment, one must first define a Release or Debug Run configuration. For that, it is required to have an ULTIMET ADVANCED connected to your PC.

19.3.5.1.1Create a Debug Run configuration

1. Select the Debug button and choose “Debug Configurations…”

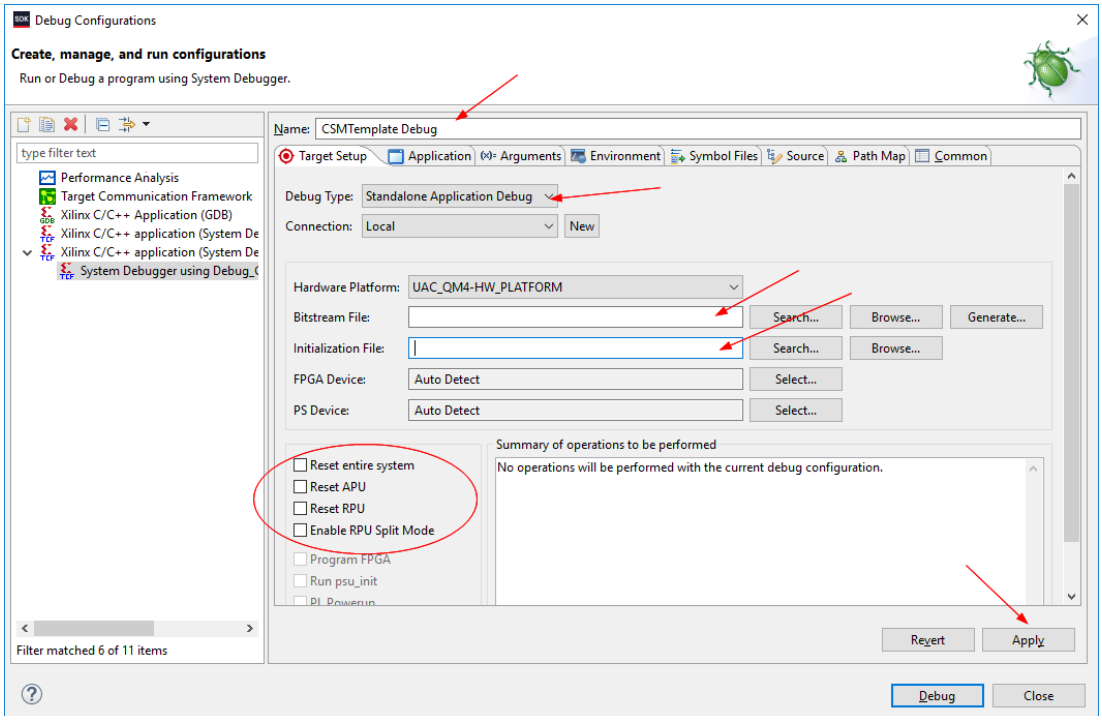


2. Select “C/C++ application (System Debugger)” and press the “New” button

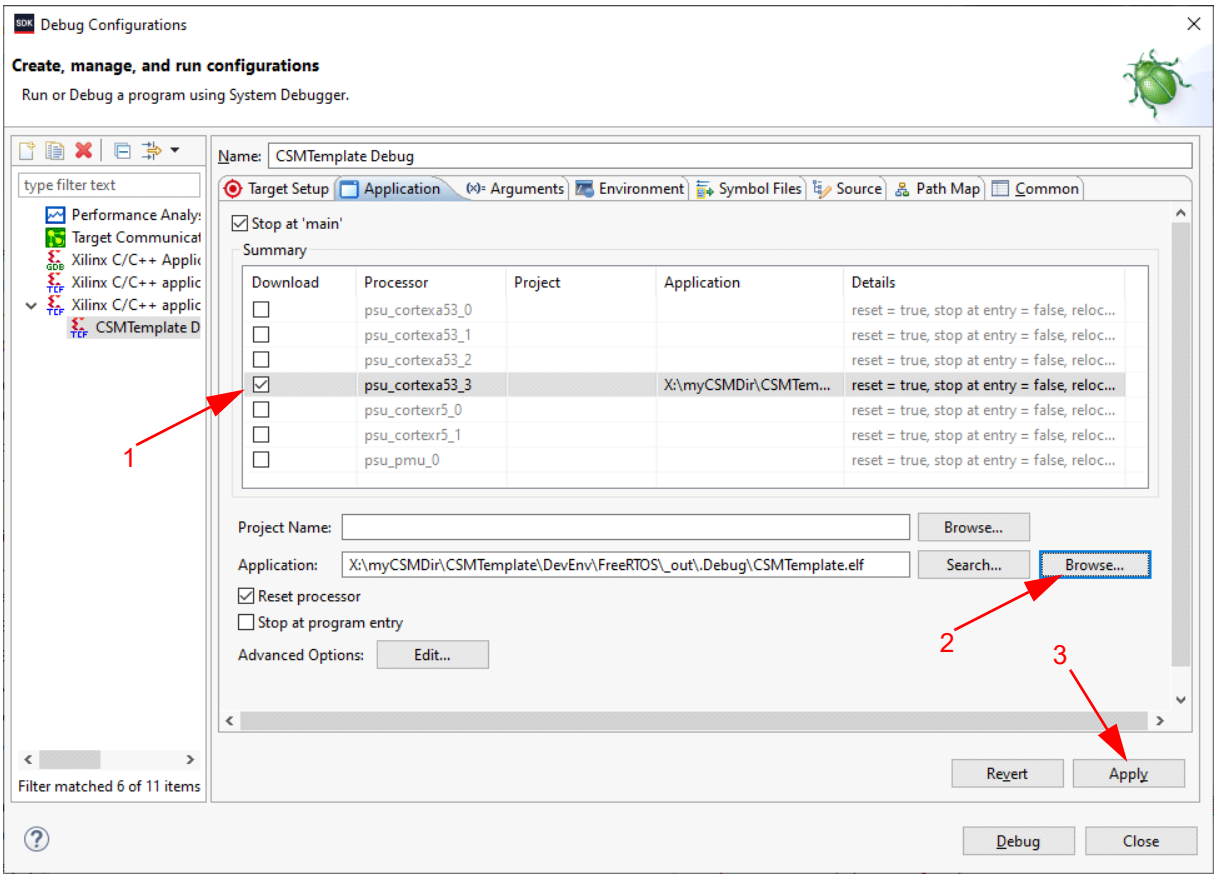


3. Change the “Name” of the configuration, here “CSMTemplate Debug” is used
4. Set “Debug Type” as “Standalone Application Debug”
5. Clear “Bitstream File”
6. Clear “Initialization File”

7. Check that nothing is enabled and apply the modifications

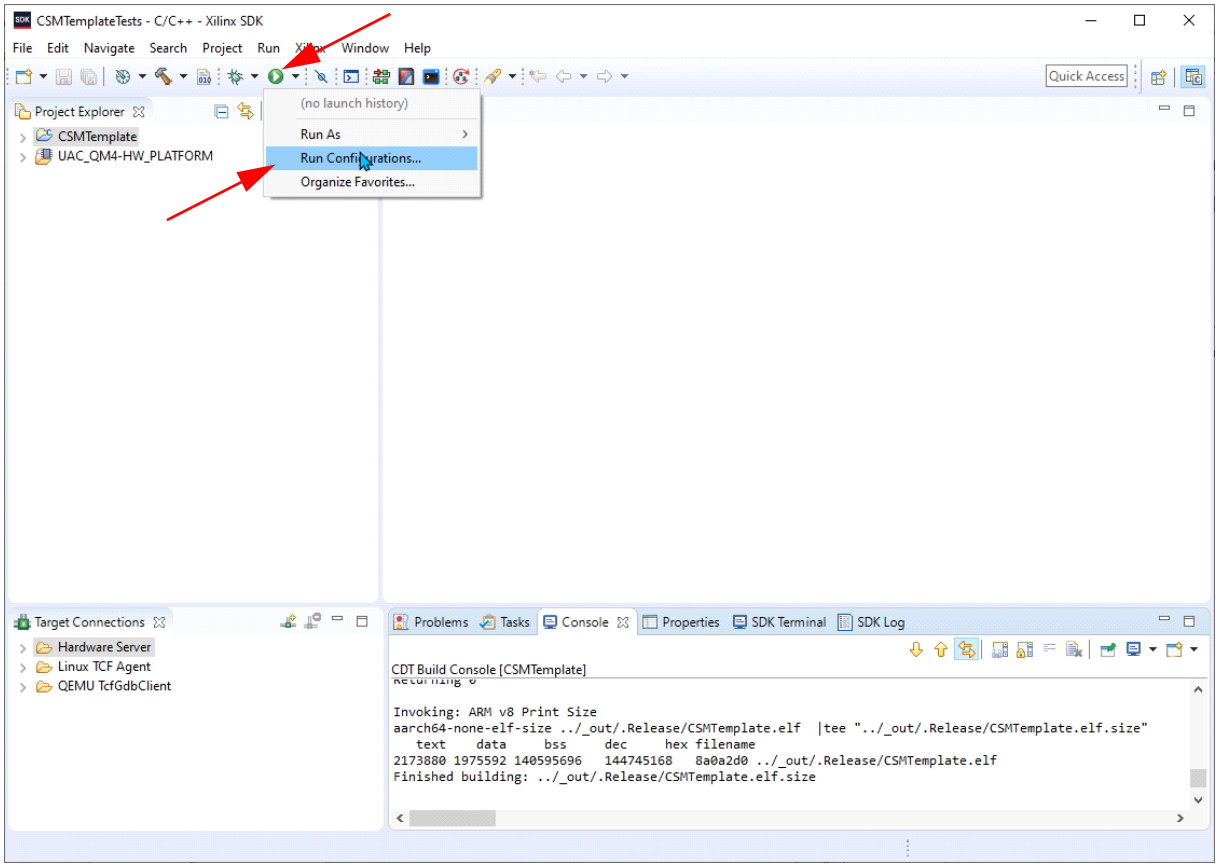


8. Go to the “Application” tab. The following actions **must be done in the order specified below:**
   1. Select “psu_cortex53_3” core
   2. Use “browse…” to enter the location of the application. The .elf file to select – CSMTemplate.elf - will be in …\myCSMDir\CSMTemplate\DevEnv\FreeRTOS_out\.Debug in the directory where you have unpacked the CSMTemplate package.
   3. Apply the modifications

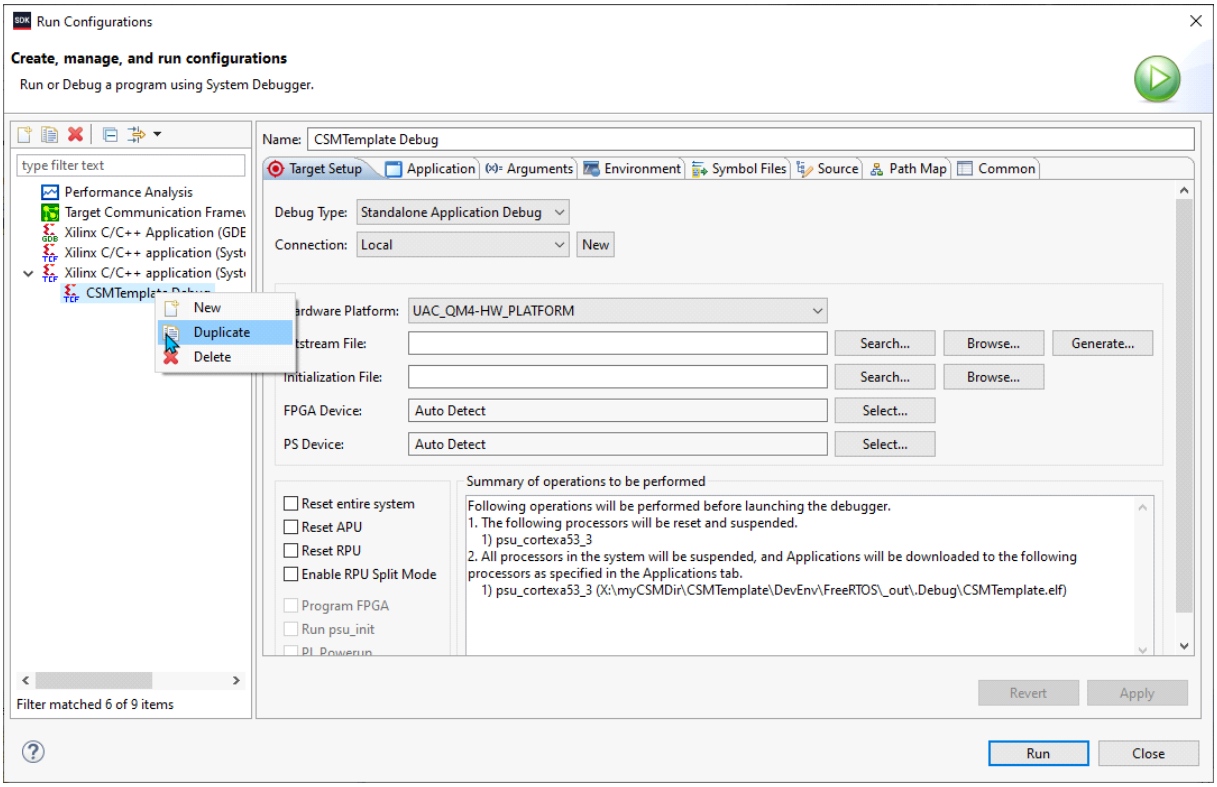


19.3.5.1.2Create a Release Run configuration

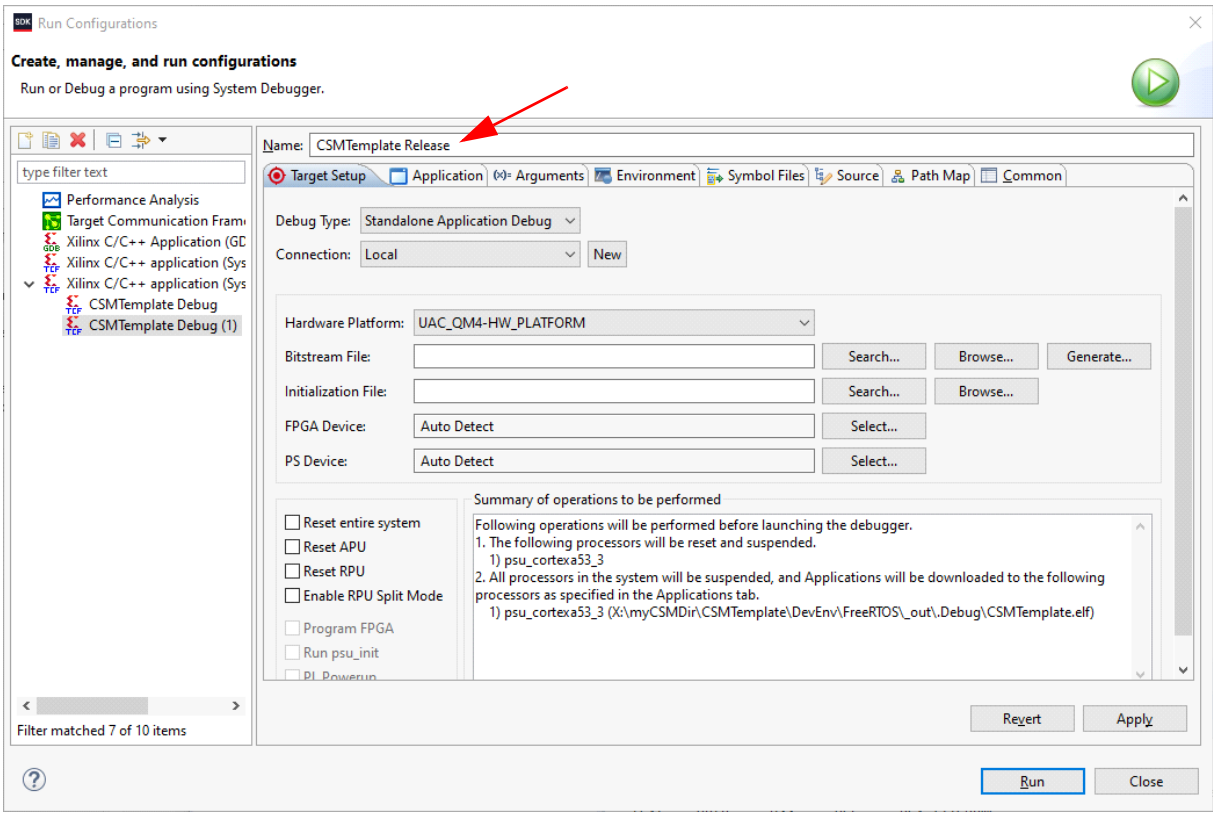
1. Select the “Run” button and choose “Run Configurations…”.



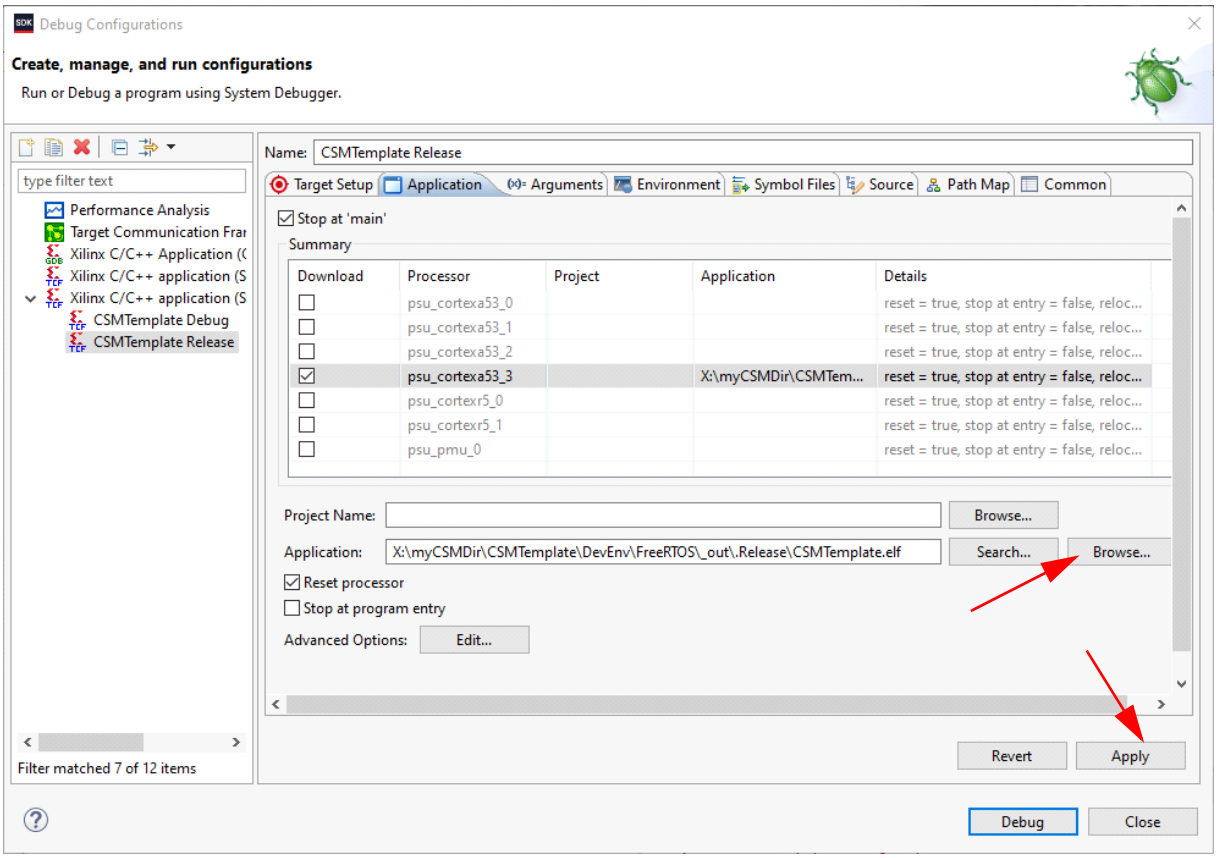
2. This time, one can duplicate the “CSMTemplate Debug” configuration.



3. On the “Target Setup” tab, change the “Name” of the configuration, e.g. “CSMTemplate Release”.



4. Switch to the “Application” tab and enter the location of the Release application with the “Browse…” button. For the Release configuration, the application will be in …\myCSMDir\CSMTemplate\DevEnv\FreeRTOS_out\.Release in the directory where you have unpacked the CSMTemplate package. Finally, “Apply” the changes.



Please remember that a “Released” version of a CSM does not have any standard Input/Output.

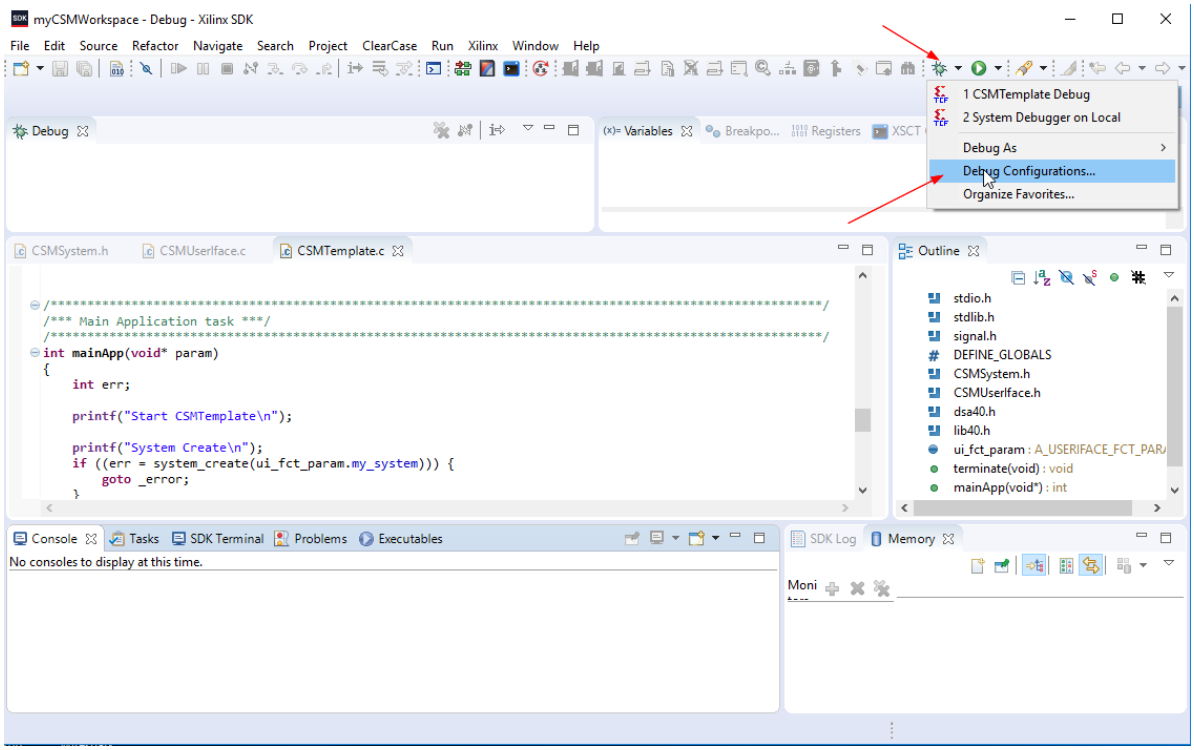
19.3.5.1.3Debug the CSM

By default, the CSM template is configured to run with one AccurET (2 axes numbered 0 and 1). This can easily be changed, refer to §20.1.2 for more information. Once a Debug configuration has been created, one can debug the CSM. For this, do not forget to connect first the ULTIMET ADVANCED to your PC with the JTAG cable. Then:

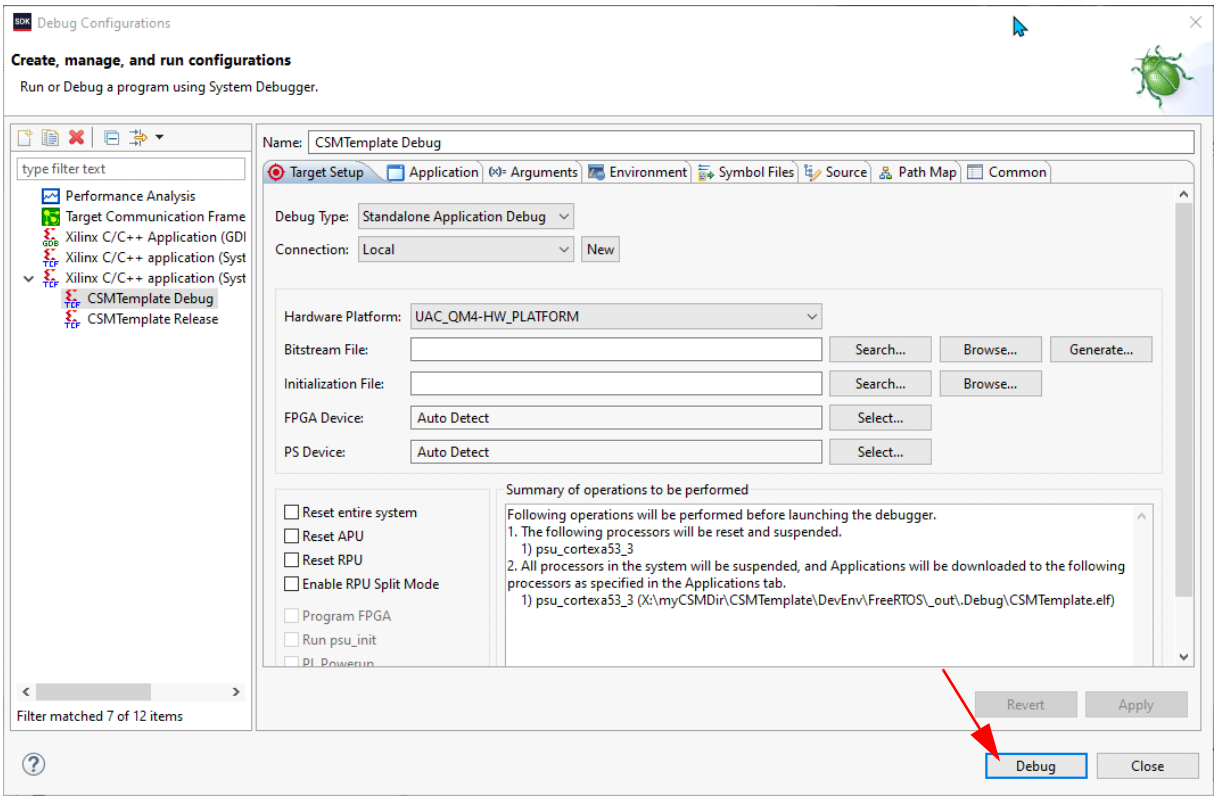
1. Switch on the ULTIMET ADVANCED.

2. Debug the CSM by
- Pressing the “Debug” button
- Choosing the Debug configuration.

Remark: As soon as a configuration is selected once, it is directly available in the list as in the example below. It is no longer required to enter in the “Debug Configurations…” panel.

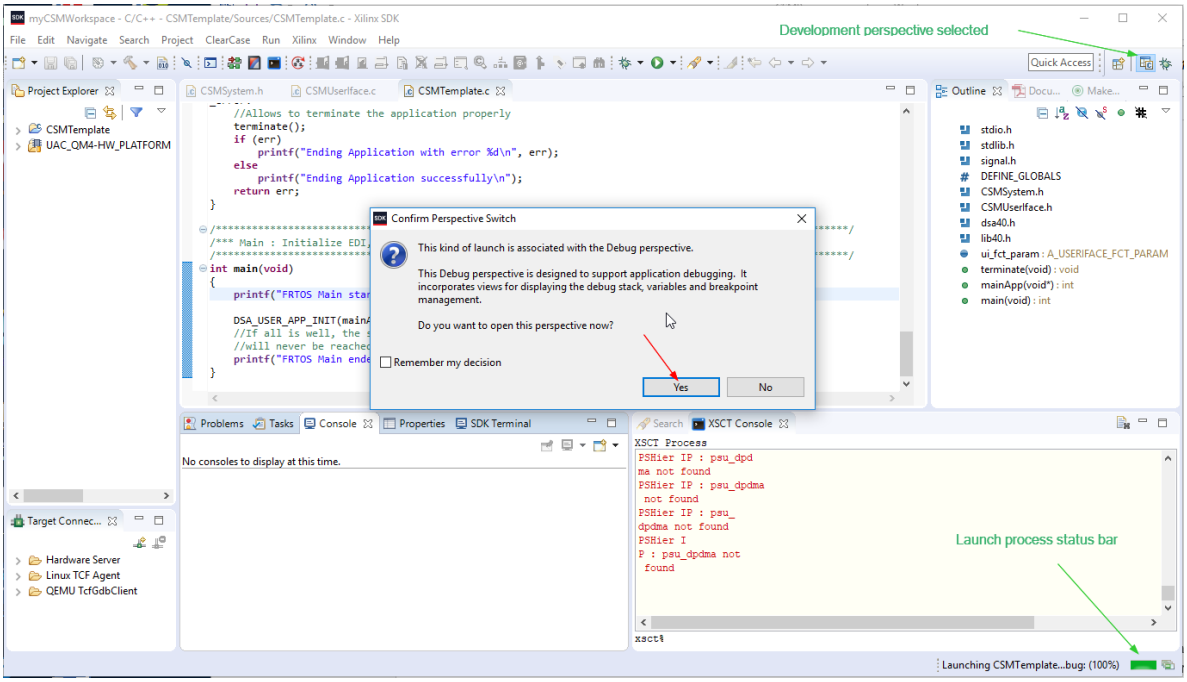


3. Press the Debug button

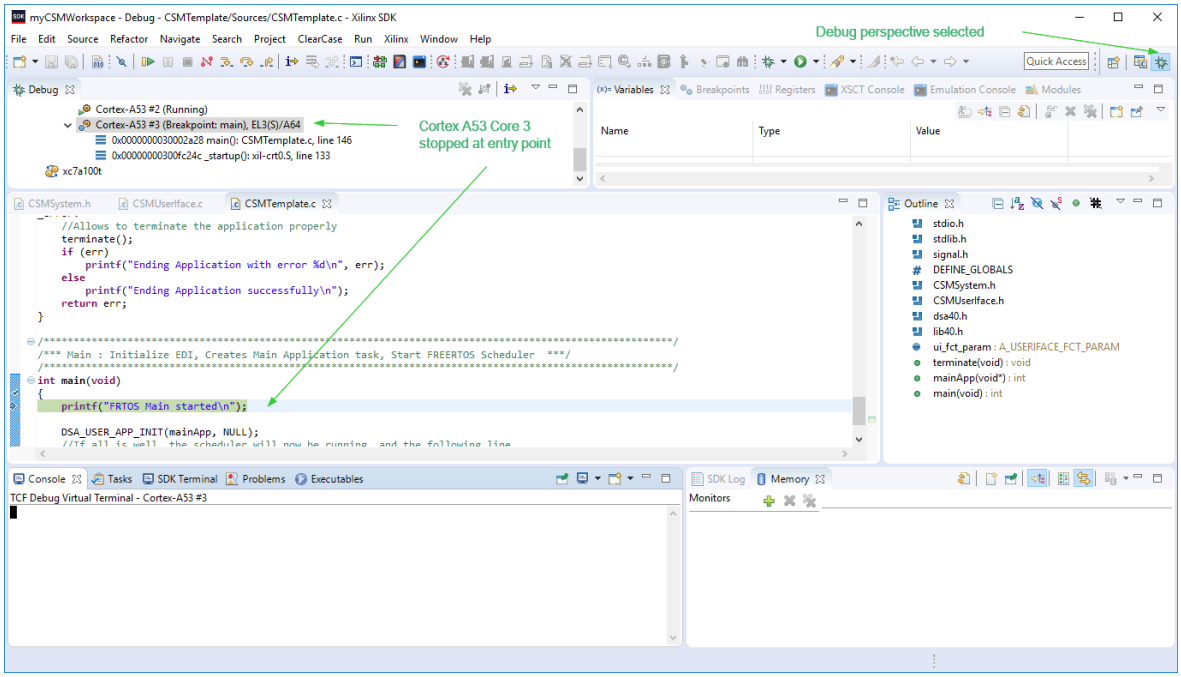


4. The SDK starts now to download the CSM into the target (this can take between 15 to 30 seconds). One can follow the progress on the status bar.

5. If the SDK is the Development perspective, it will inform the user that the SDK will switch into the Debug perspective. Simply answer “Yes”.



6. Once the CSM has been downloaded, the SDK switches into Debug perspective. The Debug is started and the CSM stops at the entry point.



7. One can now start debugging the CSM using Run, Step into, Step over, etc.

19.3.5.2 Execution from the host application

To be able to start a CSM from a host application, the CSM must be present on the micro-SD card. The CSM file is stored into the reserved directory of the micro-SD card when downloaded on the UltimET Advanced. The CSM is downloaded on the UltimET Advanced using the ComET tool. The downloaded file is renamed CSM1.bin by ComET as expected by the firmware.

Refer to §22.1 to see how to start a CSM.

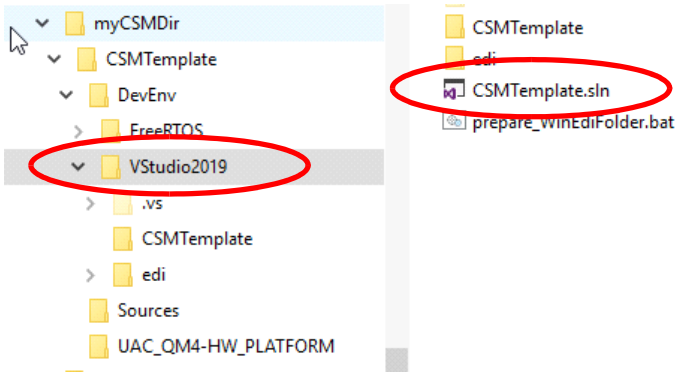
19.4 CSM Template Visual Studio project

19.4.1 Installing

Refer to the Microsoft indications to install your Visual Studio environment. The CSMTemplate itself needs no specific installation.

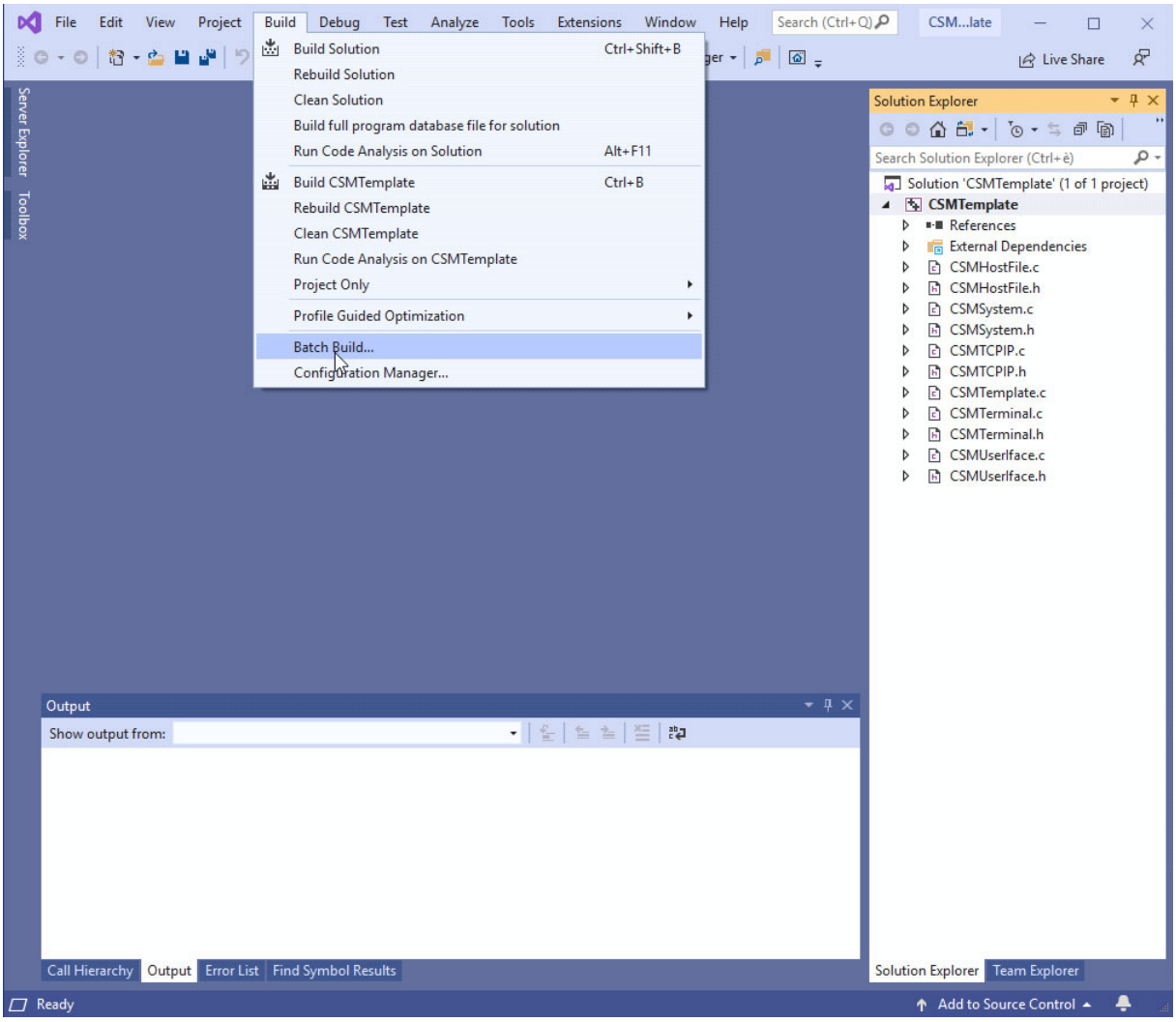
For this development environment, you will find the solution and project all set up to be compiled under Windows, with the links to the same source files as for FreeRTOS. The EDI Windows libraries are there, to be linked with the CSM.

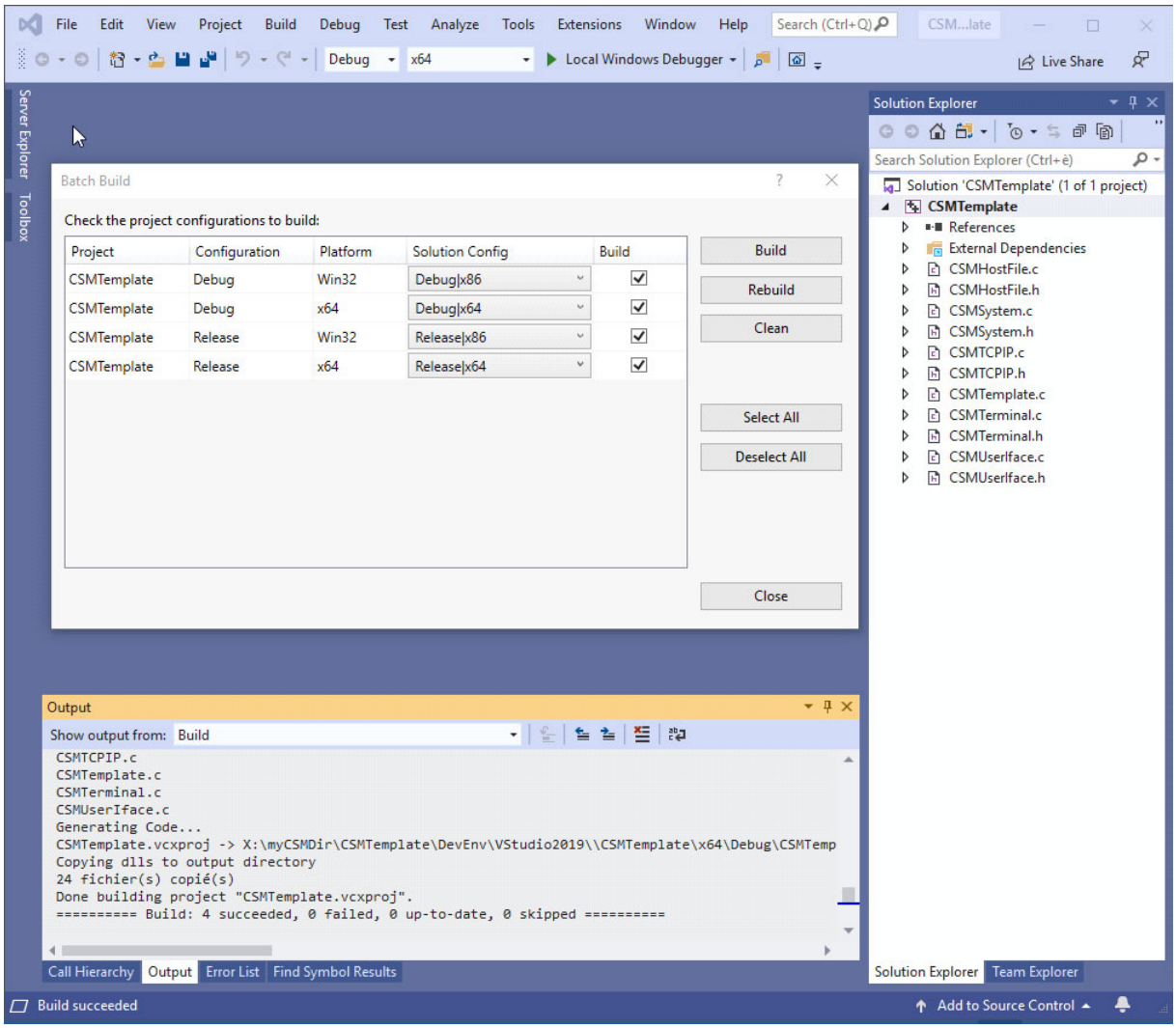
The CSMTemplate VisualStudio solution can be found there:



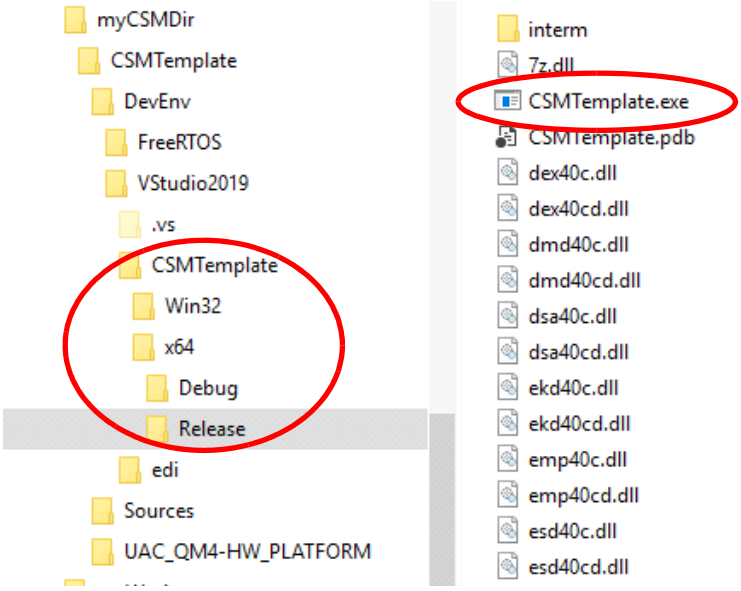
19.4.2 Building

The solution contains the project and configurations ready to run in release and debug mode. You can build the CSMTemplate for a 32–bit architecture and a 64-bit architecture:





The executable file, with all the dlls required to run it are found here:



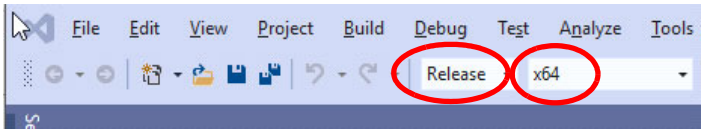
19.4.3 Running

19.4.3.1 Release version

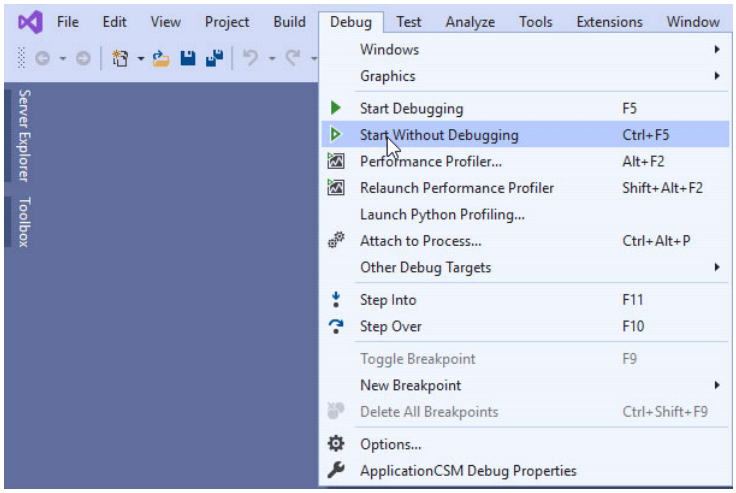
19.4.3.1.1From Visual Studio

You can run the release directly from Visual Studio:

1. Select the architecture you want to run:



2. Select “Debug/Start Without Debugging”:



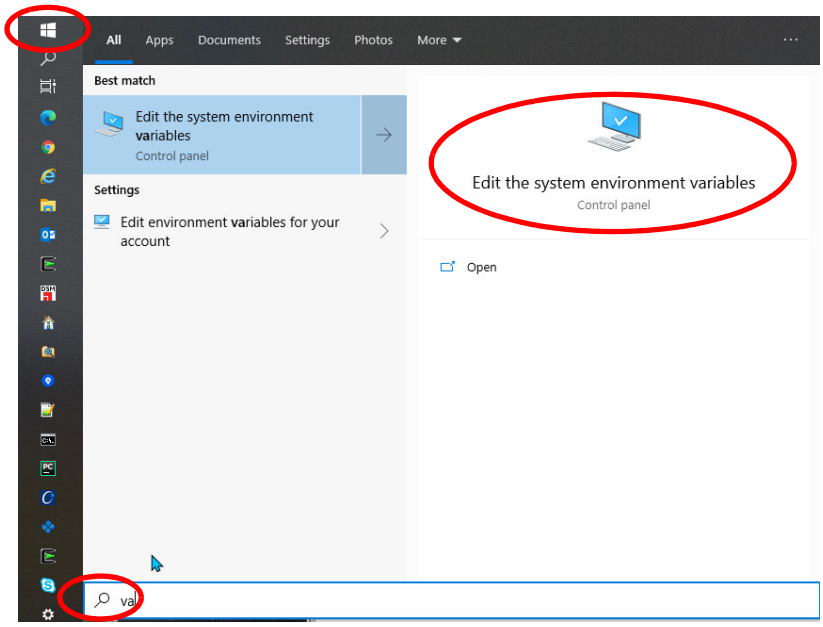
19.4.3.1.2Standalone

To run the CSMTemplate release version:
- Either double-clink on the .exe file,
- or in a command window go to the output directory and type CSMTemplate.exe,
- alternately, you can copy the CSMTemplate.exe where you like and the dlls where you like also as long as:
    - either the dlls are in the same directory as the executable files
    - or the path to the dlls is in the PATH environment variable.

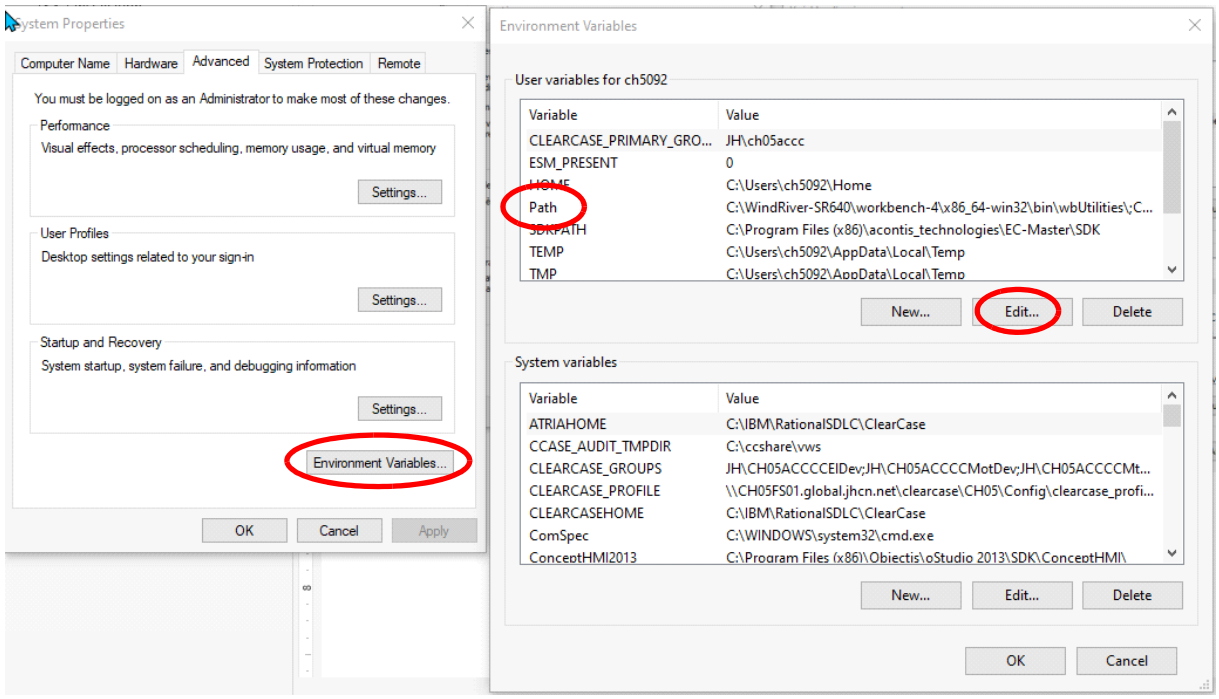
The dlls are provided in the same directory as the application output as shown above. If you move the dlls elsewhere, you can add the path to the new location in the Windows environment variable setting menu:

1. Click on the “Windows” icon in the windows task bar.

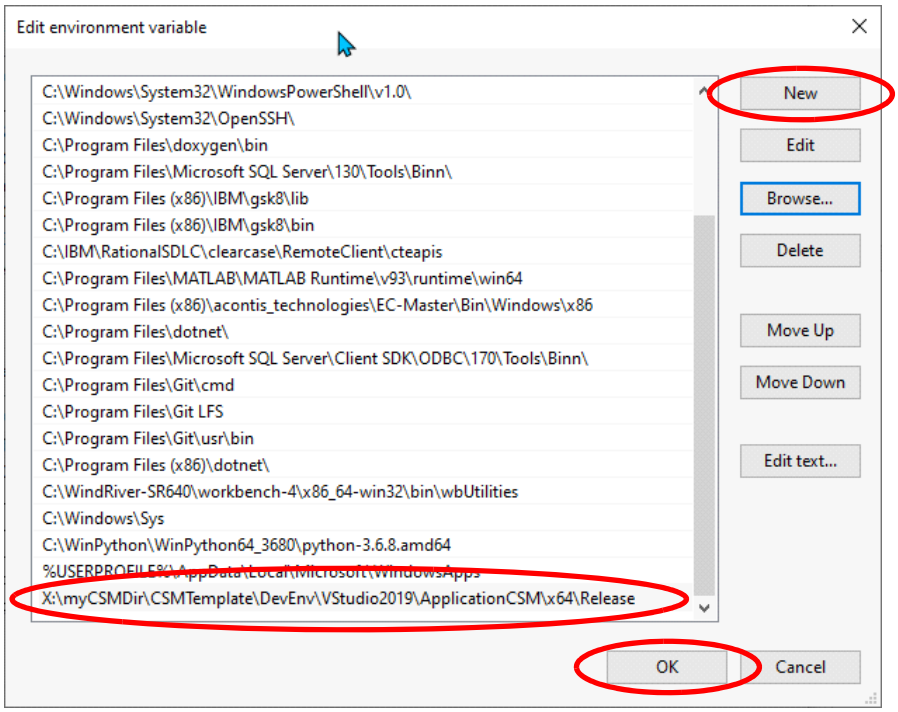
2. Start typing “var …” and then return:



3. Select “Environment Variables” and then in the “users” section, select “Path” and “Edit:



4. Click on “New” and type in the path to where you have put the dlls:

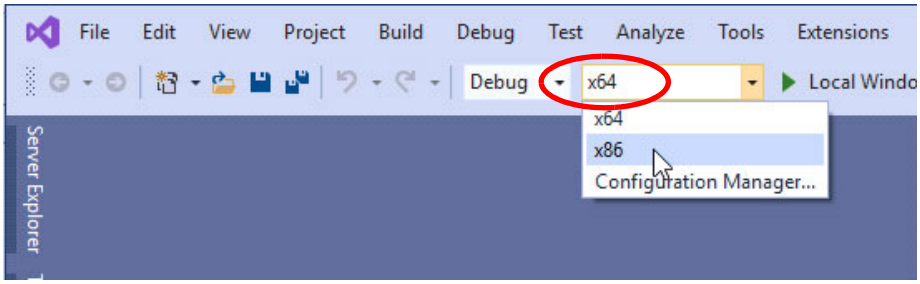


5. Click “OK” to validate.

19.4.3.2 Debug version

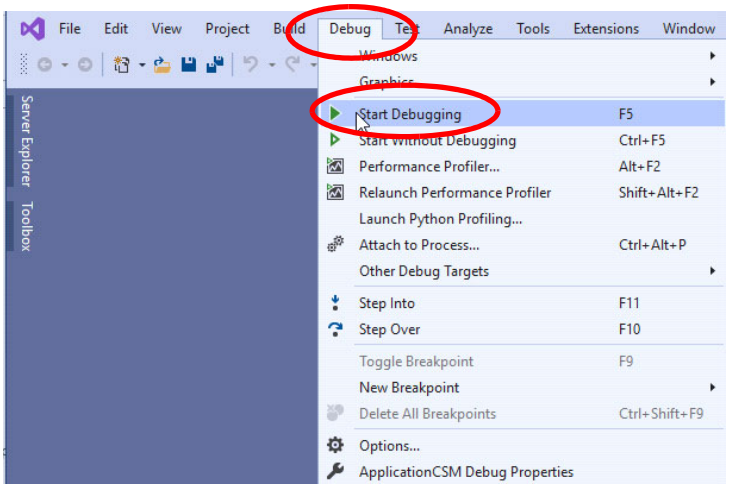
You run the debug version from the Visual Studio environment.

1. First select the architecture you want to debug:



Remark: The 32-bit architecture is called x86 by Visual Studio.

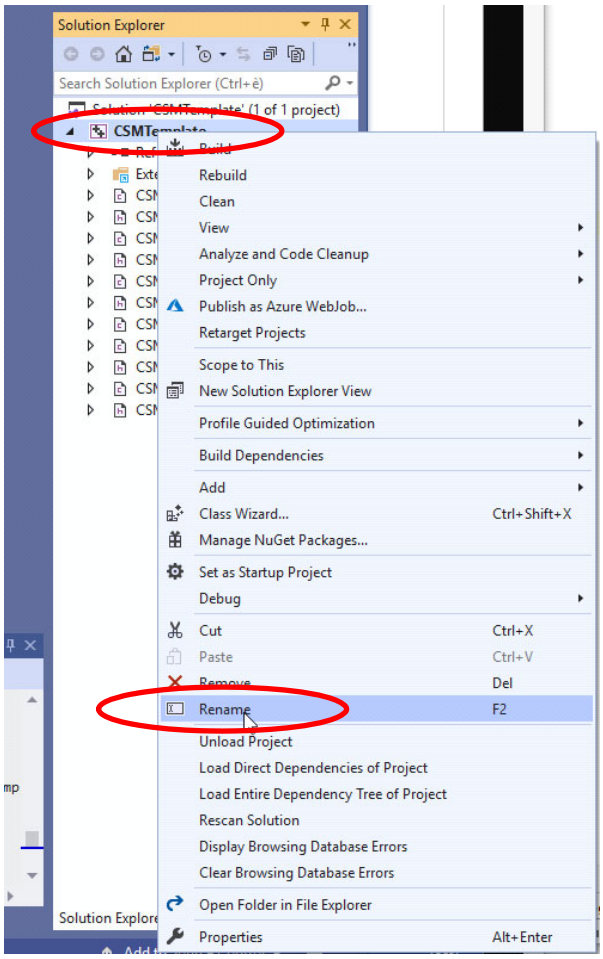
2. Either type “F5” or select “Debug/Start Debugging:



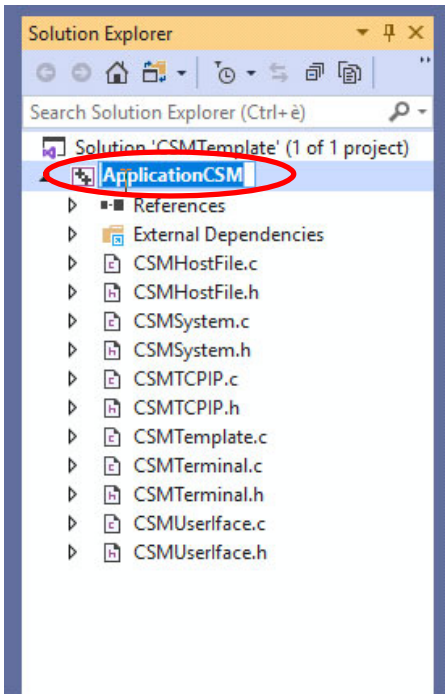
19.4.4 Renaming

In this environment also, you can rename the CSMTemplate project to your liking.

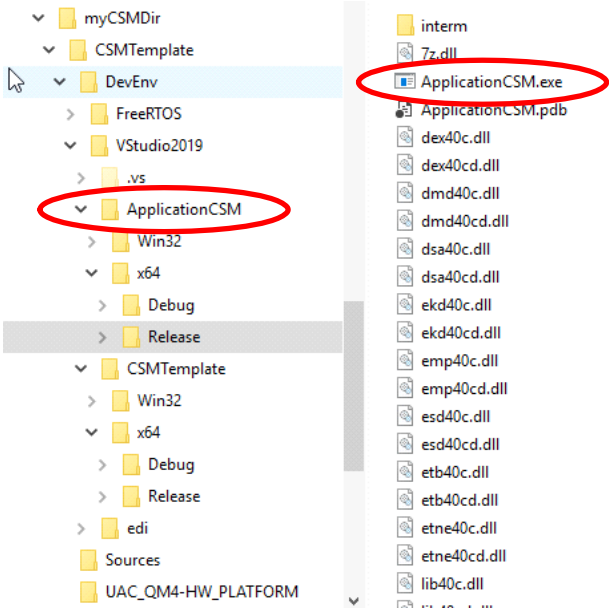
1. Right-click on the CSMTemplate project name and selecting “Rename”:



2. And typing in a new name for the project:



3. After a new rebuild, the new executable will be found in the new project output:



20 CSM template

20.1 Structure

The CSM template source code contains several components:

Component	Description

CSMTemplate.c	• Contains the C `main` function that
	  • Initializes EDI by creating a thread that runs the "`mainApp`" user function and starts the scheduler
	• Contains the C `mainApp` function. This is the main function of the user application that
	  • Initializes the CSM by creating the ETEL devices and groups of devices
	  • Opens the communication to the created devices
	  • Starts the user interface

CSMSystem.*	• Contains the definition of devices (axes and master) that will be accessed by the CSM
	• Contains the definition of device groups that will be accessed by the CSM
	• Contains functions allowing to
	  • Create, open, close and destroy the axes objects
	• Contains examples to:
	  • Reset the error on all connected devices
	  • Read/write/compare register values on all devices
	  • Start/Stop an error manager

CSMUserIface.*	• Contains the functions that start/stop the user interface
	• Contains the high-level entry point functions that can be accessed by the user interface
	  • Either from a Host application via a *JMP function
	  • Or from the Xilinx SDK console user interface (development phase)

CSMTerminal.*	Terminal allowing to enter ETEL-like commands as in the ComET terminal. This Terminal only runs when the USER INPUT is enabled (development phase using Xilinx SDK environment).
CSMHostFile.*	Example showing how to write/read files to/from the user SD card.

CSMTCPIP.*	• Contains a small TCP/IP "echo" client/server example composed of:
	  • A simple Server listening and accepting TCP/IP connection requests from Clients.
	  • The connection-dedicated server that is created on a Client connection that echoes back all messages sent by the Client.
	  • A Client establishing connection to the hot Server, sending messages, waiting for the returned echos and displaying both sent and received messages.
	Special attention is required to the way a Client disconnection is managed:
	• It is mandatory to set the socket option `SO_RCVTIMEO`. This allows the connection-dedicated server to detect that its Client has disconnected from the socket and "terminate" itself gracefully (otherwise the thread associated to this connection remains alive).

lscript.ld	This special file is used during the link of the CSMTemplate. It contains several mandatory definitions about the memory layout of the CSM. **You must not modify this file, otherwise the whole system may become instable.**

20.1.1 User interface (CSMUserIface.*)

This component handles the user interface. The user interface is the way that the CSM exports its internal functionalities to the external world. The external world can be:
- The Xilinx SDK console (during development phase).
- Any host application (customer application, ComET, …) (during production phase).

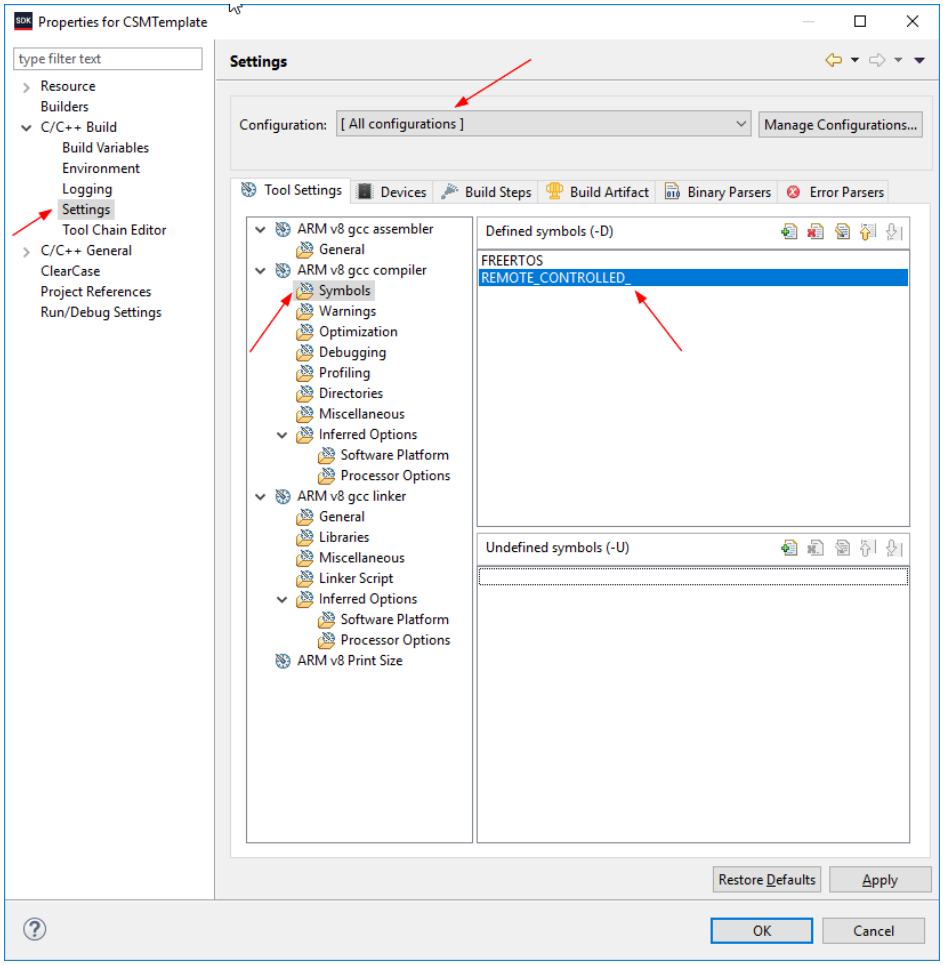
20.1.1.1 User interface for the Xilinx SDK console

In this mode, the CSM User Interface is limited to:
- Displaying the list of available CSM high-level functions.
- Allowing the user to choose, with the keyboard, the high-level function to execute.
- Executing the specified high-level function.

Remark: To compile the project `CSMTemplate` for this mode, be sure that your project does NOT define the `REMOTE_CONTROLLED` pre-processor directive.

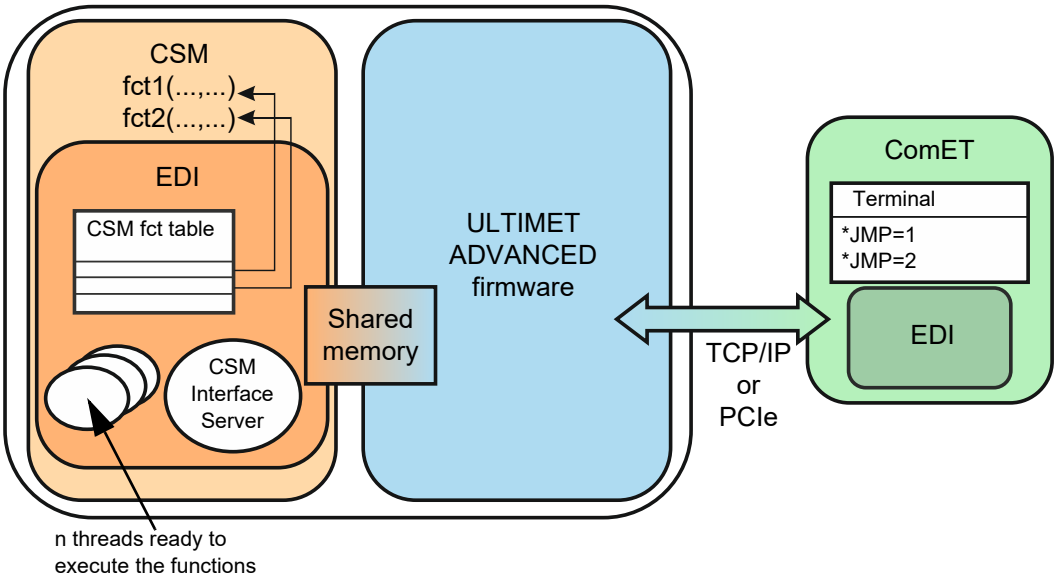
The directive `REMOTE_CONTROLLED` can be modified like this

- Right click on `CSMTemplate` project
- Choose “Properties”
- Go into “C/C++ Build” → “Settings panel”
- In “Tool Setting” tab, choose option “ARM v8 gcc compiler” → “Symbols”
- Be sure you are setting the symbols for “All configurations”
- The symbol `REMOTE_CONTROLLED` must not be set. A suggestion is to change it into `REMOTE_CONTROLLED_` (this allows to remember the definition for a future use).
    - With the `REMOTE_CONTROLLED` directive defined, the project will be compiled without user input (final production environment).
    - Removing the `REMOTE_CONTROLLED` directive (or renaming it to `REMOTE_CONTROLLED_`), the project will be compiled with user input (development phase environment).



20.1.1.2 User interface for an Application environment

In this mode, the CSM should be able to receive some commands coming from a host application through the ULTIMET ADVANCED. For that, the CSM has to configure and start an ULTIMET ADVANCED CSM Command server.



This ULTIMET ADVANCED CSM command interface server can be configured by:
- Specifying the number of threads (up to 8) ready to run CSM high-level functions. To save resources, it is recommended to minimize the number of threads initiated.
- Specifying the number of available high-level functions.
- Attaching each CSM high-level function to the server.

Refer to function `user_iface_start()` in the `CSMUserIface.c` source file for an example of the creation, configuration and startup of an ULTIMET ADVANCED CSM command interface server.

When started, the ULTIMET ADVANCED CSM command interface server will launch:
- A Command interface thread which will accept the commands coming from the ULTIMET ADVANCED. Actually, the accepted commands are:
```
*JMP=<fctNb>,<threadNb>  // Start a CSM high level function
*END=<threadNb>          // Stop a thread executing a CSM high level function
```
- Up to 8 threads, ready to run CSM high-level functions.

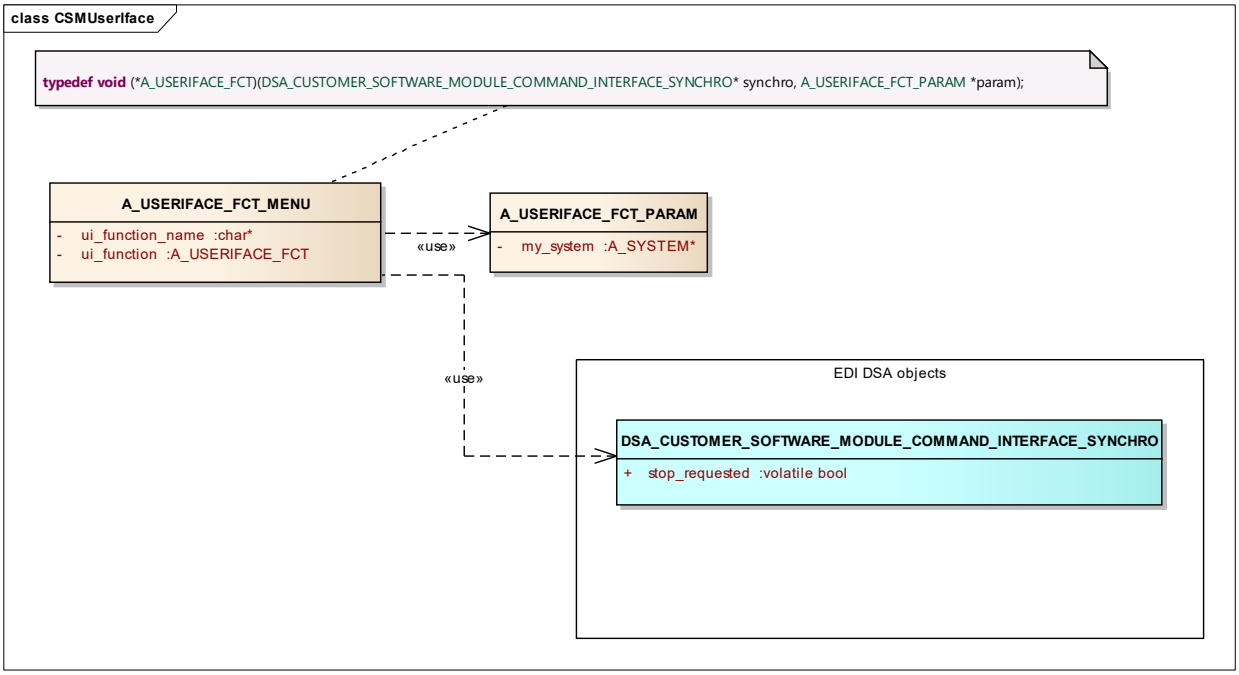
To compile the `CSMTemplate` for the final application, the `-D REMOTE_CONTROLLED` pre-processor directive must be defined (refer to §20.1.1.1 for the procedure).

Two monitoring registers report the CSM activity:
- ***M97** indicates which threads are running.
- ***M95** indicates what CSM function is running in each thread.

M	Name	Comment
***M97**	CSM threads running	Mask of the threads that have a high level function running. Bit `threadNb-1` set to 1 by the *JMP command and reset to 0 by the *END command
***M95**	CSM high level functions running	Each depth corresponds to a thread (`threadNb > -1`), and the value at that depth is the index of the CSM high level function <fctNb> that was started in that thread by the *JMP=<fctNb>, <threadNb> command, 0 otherwise.

Remark: Starting the ULTIMET ADVANCED CSM command interface server is not mandatory. However, if the server is not initiated, a host application will not be able to start/stop CSM high-level functions.

Here is the UML representation of the objects defined in `CSMUserIface.c`. All the classes are defined as C-structure or C-typedef.



- **`A_USERIFACE_FCT_MENU` structure**

This object allows storing a CSM high-level function that can be called by:

- The user interface display on the Xilinx SDK console (development phase).
- `*JMP`/`*END` commands sent to ULTIMET ADVANCED by a host application and forwarded to CSM through the ULTIMET ADVANCED CSM command interface server.

The `user_iface_start` function will create a table of this kind of object, allowing the CSM to "export" its high-level functions. All functions in this table will be accessible via the Xilinx SDK console or the ULTIMET ADVANCED CSM command interface server.

Refer to function `user_iface_start()` in the `CSMUserIface.c` source file for example.

- **`A_USERIFACE_FCT` type**

Defines the prototype of a high-level function. All high-level functions should respect this prototype. A high level function does not return any value and takes two parameters of the following type:

- `A_USERIFACE_FCT_PARAM` structure

This structure is the main high-level function parameter. As each high-level function should respect the `A_USERIFACE_FCT` prototype, it should contain everything that is needed. The user can modify this structure by adding each object reference it needs for his application.

- `DSA_CUSTOMER_SOFTWARE_MODULE_COMMAND_INTERFACE_SYNCHRO` structure

This structure allows the ULTIMET ADVANCED CSM server to stop a high-level function. It is often required that a high-level function is programmed as an endless loop, realizing some supervision tasks.

To be able to stop such endless function, such function should regularly check if the ULTIMET ADVANCED requests to stop it. Therefore, a function executing an endless loop should be programmed as follow:

```
void endless_loop(DSA_CUSTOMER_SOFTWARE_MODULE_COMMAND_INTERFACE_SYNCHRO*
                  synchro, A_USERIFACE_FCT_PARAM* ui_fct_param)
{
    int err;
    while (!synchro->stop_requested) {
        ...
    }
}
```

Refer to function `ui_loop_all_write_read_registers()` in the `CSMUserIface.c` source file for an example of an endless interruptible function.

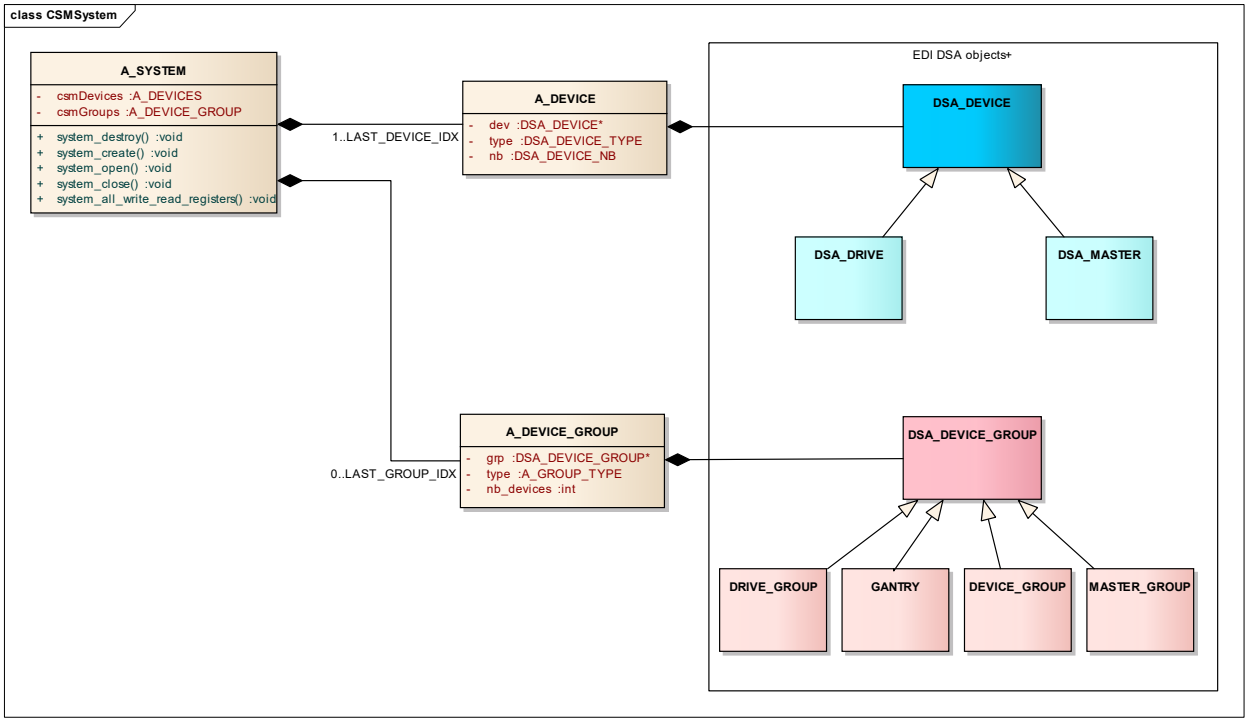
Remark: **If the user does not program an endless function as the example above, the function cannot be stopped except by destroying the CSM.**
This is particularly dangerous in the case of urgency which request urgent commands (!*HLO, !*HLB, !*HLT). Refer to §15.4.4.4 for more information.

20.1.2 System (CSMSystem.*)

This component handles the system objects. These objects are:
- The devices, such as AccurET position controllers or UltimET.
- The devices' groups.

This component contains all functions required for accessing the devices (through EDI). Some of these functions can be made available to the external world and act as CSM high level functions as described in §20.1.1.

Here is the UML representation of the objects defined in `CSMSystem.c`. All the classes are defined as C-structure or C-enumeration.



- **`A_SYSTEM` structure**

This is the main object of the CSM. This object contains:
  - An array containing all system single devices. The devices can be accessed by their index as enumerated in the `A_DEVICE_IDX` C-enumeration.
  - An array containing all device groups defined in the system. The device groups can be accessed by index enumerated in the `A_GROUP_IDX` C-enumeration.

- **`A_DEVICE` structure**

This kind of object represents a single device. It stores:
  - The underlying `DSA_DEVICE` object (the object required by DSA functions).
  - The device type, which can be:
    - `MASTER` for the ULTIMET ADVANCED itself.
    - `DRIVE` for an AccurET position controller.
  - The device number

Remark: There is no need to modify this data structure.

- **`A_DEVICE_GROUP` structure**

This kind of object represents a group of devices. It stores:
- The underlying `DSA_DEVICE_GROUP` object (the object required by DSA functions)
- The group type, which can be:
  - `DRIVE_GROUP`: for a group of AccurET only.
  - `GANTRY`: for a group of 2 AccurETs configured in gantry level 1.
  - `MASTER_GROUP`: for a group of ULTIMET ADVANCED (not supported today).
  - `DEVICE_GROUP`: for a group mixing AccurET and ULTIMET ADVANCED.
  - The number of devices.
  - An array of device indexes: allowing to access the device in `A_SYSTEM::csmDevices` table.

By default, the `CSMTemplate` connects to 3 `CSMDevice` objects (2 axes and the ULTIMET ADVANCED). It is however possible to connect up to 7 `CSMDevice` objects (6 axes and the ULTIMET ADVANCED) by re-defining the following pre-processor definitions:

Definitions	Action
#define FOUR_AXES 1	Four axes are defined.
#define SIX_AXES 1	Six axes are defined.

Remark: If both `FOUR_AXES` and `SIX_AXES` definitions are commented, only 2 axes will be defined. If both are defined, six axes are available.

List of pre-defined objects in the `CSMTemplate` (with all devices enabled):

`A_DEVICE` objects	Device Type	Device Number	Device
csmDevices[AXISX1_IDX = 0]	DRIVE	0	X1
csmDevices[AXISX2_IDX = 1]	DRIVE	1	X2
csmDevices[AXISY1_IDX = 2]	DRIVE	2	Y1
csmDevices[AXISY2_IDX = 3]	DRIVE	3	Y2
csmDevices[AXISZ_IDX = 4]	DRIVE	4	Z
csmDevices[AXIST_IDX = 5]	DRIVE	5	Theta
csmDevices[ULTIMET_IDX = 6]	MASTER	63	UltimET

`A_DEVICE_GROUP` objects	Group Type	Devices
csmGroups[GROUPX1X2_IDX = 0]	GANTRY	X1, X2
csmGroups[GROUPX1X2MASTER_IDX = 1]	DEVICE_GROUP	X1, X2, UltimET
csmGroups[GROUPY1Y2_IDX = 2]	GANTRY	Y1, Y2
csmGroups[GROUPXYT_IDX = 3]	DRIVE_GROUP	X1, X2, Y1, Y2, Theta
csmGroups[GROUPALLAXIS_IDX = 4]	DRIVE_GROUP	X1, X2, Y1, Y2, Z, Theta
csmGroups[GROUPALL_IDX = 5]	DEVICE_GROUP	X1, X2, Y1, Y2, Z, Theta, UltimET

Remark: To define another system structure, the user has to modify the `CSMSystem.h` file accordingly.

The CSMSystem functions:
- `system_create`, `system_open`, `system_close` and `system_destroy`

will create, open, close and destroy the defined objects automatically.

20.1.2.1 Example of adding a new drive in `CSMSystem.h`

This example shows how to:
- Add a new drive named L with axis number 6.
- Add a new group containing X1, X2 and L axes.
- Modifying the ALL_AXIS and all groups to refer also to axis L.

Edit `CSMsystem.h` as follows:

Remark: All modifications are highlighted in red.

- Uncomment the pre-processing macros:

```
#define FOUR_AXES 1     // Uncomment if you want the CSM running with 4 axes
#define SEVEN_AXES 1    // Uncomment if you want the CSM running with 7 axes
```

In A_DEVICE_NB enumeration, add axis L axis number:

```
typedef enum {
    AXISX1_NB     = 0,
    AXISX2_NB     = 1,
#ifdef FOUR_AXES
    AXISY1_NB     = 2,
    AXISY2_NB     = 3,
#endif
#ifdef SEVEN_AXES
    AXISZ_NB      = 4,
    AXIST_NB      = 5,
    AXISL_NB      = 6,
#endif
    ULTIMET_NB    = 63
} A_DEVICE_NB;
```

- In `A_DEVICE_IDX` enumeration, add the L axis index:

```
typedef enum {
    AXISX1_IDX,      // Axis X1 Device index in 'my_system.csmDevices' table
    AXISX2_IDX,      // Axis X2 Device index in 'my_system.csmDevices' table
#if defined FOUR_AXES || defined SEVEN_AXES
    AXISY1_IDX,      // Axis Y1 Device index in 'my_system.csmDevices' table
    AXISY2_IDX,      // Axis Y2 Device index in 'my_system.csmDevices' table
#endif
#ifdef SEVEN_AXES
    AXISZ_IDX,       // Axis Z Device index in 'my_system.csmDevices' table
    AXIST_IDX,       // Axis T Device index in 'my_system.csmDevices' table
    AXISL_IDX,       // Axis L Device index in 'my_system.csmDevices' table
#endif
    ULTIMET_IDX,     // ULTIMET Device index in 'my_system.csmDevices' table
    LAST_DEVICE_IDX // Do not modify
} A_DEVICE_IDX;
```

- In `A_GROUP_IDX` enumeration, add the X1-X2-L group index:

```
typedef enum {
    GROUPX1X2_IDX,       // X1-X2 Group Index in 'my_system.csmGroups' table
    GROUPX1X2MASTER_IDX,// X1-X2-Master Group Index in 'my_system.groups' table
#if defined FOUR_AXES || defined SEVEN_AXES
    GROUPY1Y2_IDX,       // Y1-Y2 Group Index in 'my_system.csmGroups' table
#endif
#ifdef SEVEN_AXES
    GROUPXYT_IDX,        // X1-X2-Y1-Y2-T Group Index in 'my_system.csmGroups'
                         // table
    GROUPX1X2L_IDX,      // X1-X2-L Group Index in 'my_system.csmGroups' table
#endif
    GROUPALLAXIS_IDX,    // X1-X2-Y1-Y2-T-Z Group Index in 'my_system.csmGroups'
                         // table
    GROUPALL_IDX,        // X1-X2-Y1-Y2-T-Z-ULTIMET Group Index in
                         // 'my_system.csmGroups' table
    LAST_GROUP_IDX       // Do not modify
} A_GROUP_IDX;
```

- In `A_SYSTEM STRUCTURE`, enter the new L drive and X1X2L group objects:

```
static A_SYSTEM my_system = {
    // Initialisation of A_DEVICE table. Max of LAST_DEVICE_IDX members
    // Must be defined in the same order as A_DEVICE_IDX enumeration
    {
        {NULL, DRIVE, AXISX1_NB},
        {NULL, DRIVE, AXISX2_NB},
#if defined FOUR_AXES || defined SEVEN_AXES
```

```
            {NULL, DRIVE, AXISY1_NB},
            {NULL, DRIVE, AXISY2_NB},
    #endif
    #ifdef SEVEN_AXES
            {NULL, DRIVE, AXISZ_NB},
            {NULL, DRIVE, AXIST_NB},
            // Be careful to put the object at the index corresponding to AXISL_IDX
            {NULL, DRIVE, AXISL_NB},
    #endif
            {NULL, MASTER, ULTIMET_NB},
        },

        // Initialisation of A_DEVICE_GROUP table. Max of LAST_GROUP_IDX members
        // Must be defined in the same order as A_GROUP_IDX enumeration
        {
            // Gantry whose DSA object can be referred by
            // my_system.csmGroups[GROUPX1X2_IDX].grp
            {NULL, GANTRY, 2, {AXISX1_IDX, AXISX2_IDX}},

            // Device group whose DSA object can be referred by
            // my_system.groups[GROUPX1X2MASTER_IDX].grp
            {NULL, DEVICE_GROUP, 3, {AXISX1_IDX, AXISX2_IDX, ULTIMET_IDX}},

    #if defined SEVEN_AXES
            // Gantry whose DSA object can be referred by
            // my_system.groups[GROUPY1Y2_IDX].grp
            {NULL, GANTRY, 2, {AXISY1_IDX, AXISY2_IDX}},

            // Drive group whose DSA object can be referred by
            // my_system.groups[GROUPXYT_IDX].grp
            {NULL, DRIVE_GROUP, 5,
                {AXISX1_IDX, AXISX2_IDX, AXISY1_IDX, AXISY2_IDX, AXIST_IDX}},

            // Be careful to put the object at the index corresponding to
            // GROUPX1X2L_IDX.
            // Drive group whose DSA object can be referred by
            // my_system.groups[GROUPX1X2L_IDX].grp
            {NULL, DRIVE_GROUP, 3, {AXISX1_IDX, AXISX2_IDX, AXISL}},

            // The following group becomes now a group of 7 axes
            // Drive group whose DSA object can be referred by
            // my_system.groups[GROUPALLAXIS_IDX].grp
            {NULL, DRIVE_GROUP, 7, {AXISX1_IDX, AXISX2_IDX, AXISY1_IDX, AXISY2_IDX,
                AXIST_IDX, AXISZ_IDX, AXISL_IDX}},

            // The following group becomes now a group of 8 axes
            // Device group whose DSA object can be referred by
            // my_system.groups[GROUPALL_IDX].grp
            {NULL, DEVICE_GROUP, 8, {AXISX1_IDX, AXISX2_IDX, AXISY1_IDX, AXISY2_IDX,
                AXIST_IDX, AXISZ_IDX, AXISL_IDX, ULTIMET_IDX}}

    #elif defined FOUR_AXES
            // Gantry whose DSA object can be referred by
            // my_system.groups[GROUPY1Y2_IDX].grp
            {NULL, GANTRY, 2, {AXISY1_IDX, AXISY2_IDX}},

            // Drive group whose DSA object can be referred by
            // my_system.groups[GROUPALLAXIS_IDX].grp
            {NULL, DRIVE_GROUP, 4, {AXISX1_IDX, AXISX2_IDX, AXISY1_IDX, AXISY2_IDX}},

            // Device group whose DSA object can be referred by
            // my_system.groups[GROUPALL_IDX].grp
            {NULL, DRIVE_GROUP, 5,
                {AXISX1_IDX, AXISX2_IDX, AXISY1_IDX, AXISY2_IDX, ULTIMET_IDX}}

    #else
            // Drive group whose DSA object can be referred by
            // my_system.groups[GROUPALLAXIS_IDX].grp
            {NULL, DRIVE_GROUP, 2, {AXISX1_IDX, AXISX2_IDX}},

            // Device group whose DSA object can be referred by
```

```
            // my_system.groups[GROUPALL_IDX].grp
            {NULL, DRIVE_GROUP, 3, {AXISX1_IDX, AXISX2_IDX, ULTIMET_IDX}}
#endif
    }
}
```

Remark about closing the communication: currently, it is forbidden to close the communication with the master from the CSM. This is due to the fact that the CSM needs this communication to be able to:

- Receive commands from Host (JMP, END,...)
- Allow file transfer to user SD.

Therefore, the closing of the communication must ensure that the master is never closed. Refer to CSMSystem.c.
There is no check inside EDI ensuring that the user does not close the master communication. If the user closes this communication, neither Host communication, nor file transfer to the user SD can any longer be done in the current CSM session.

20.1.3 Terminal (CSMTerminal.*)

This component implements a basic terminal, allowing the user to input ETEL-like commands similar to the **ComET** terminal. As it requires user interaction, the use of this functionality is mainly justified during the development phase. It is available only if the CSM is compiled without the `REMOTE_CONTROLLED` pre-processor directive.

This terminal uses the EDI TRA library, which allows to translate strings containing the user input into ETEL protocol commands sent to the ULTIMET ADVANCED.

Refer to the `terminal` function in the `CSMTerminal.c` source code for an example.

Remark: This terminal does not support ISO values, only increments values are accepted.

20.1.4 Template (CSMTemplate.c)

This component contains the `main`() entry point and the `mainApp`() function of the CSM.

20.1.4.1 Start CSM

The `mainApp`() function of the CSM has to:

- Create all system objects needed by the CSM.
- Open communication to these objects.
- Create and configure the CSM user interface.
- Start the CSM user interface.

20.1.4.2 Terminate CSM

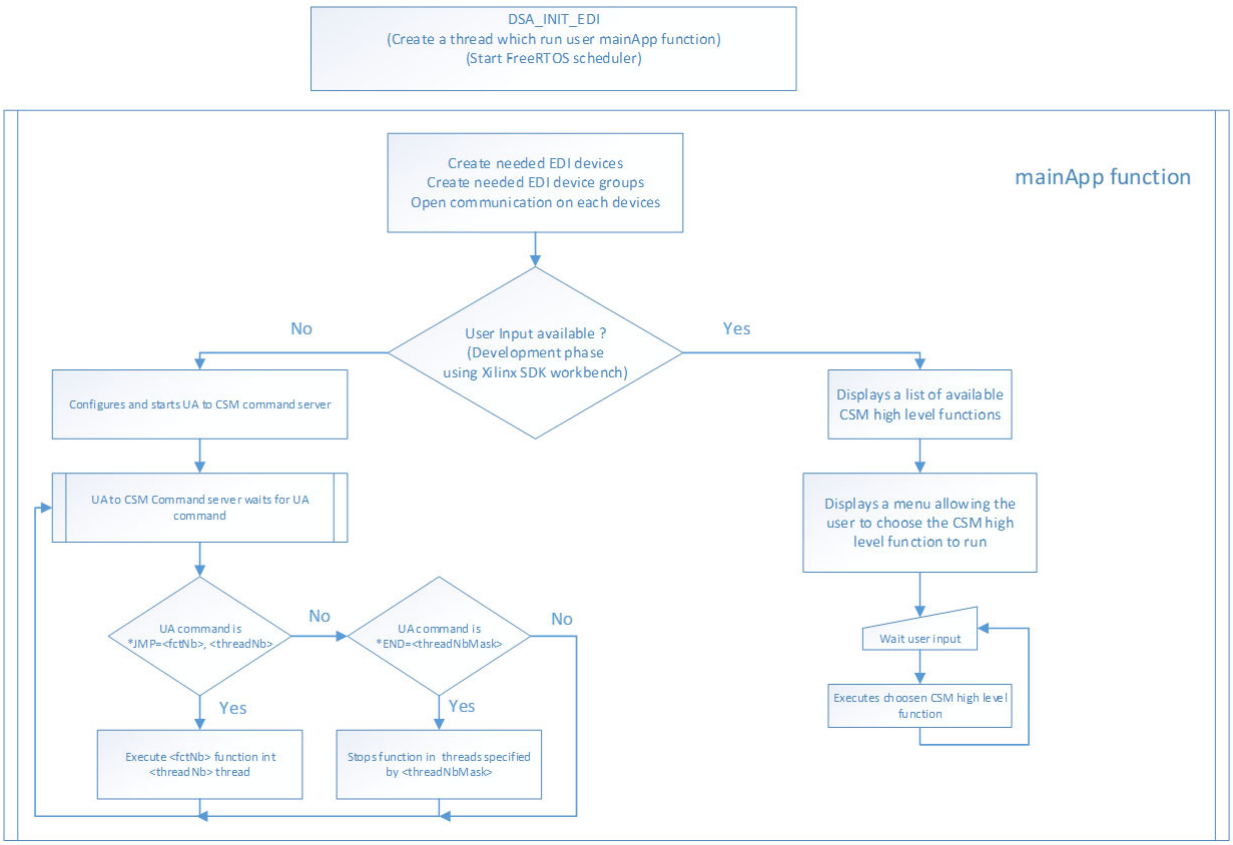
When finished, the CSM should close the communication and destroy all created objects. A CSM can terminate in many ways:

- Normal termination, i.e. execution of all instructions of the CSM.
- Host application sends a `*STOPCSM` command.
- User ends the CSM via the Xilinx SDK environment.
- Unhandled crash of the CSM (during development phase).

Remark: As Core 3 of the processor on which the CSM runs is reset when the CSM is initialized, it is not mandatory to close the communication nor destroy created objects before exiting. The ULTIMET ADVANCED handles reconnections correctly.

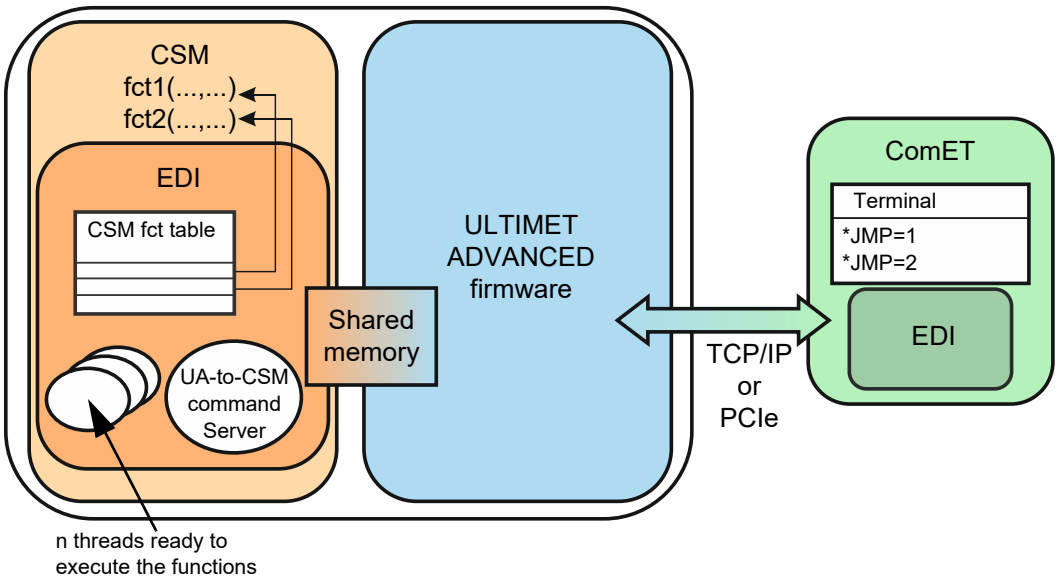
20.2 Workflow

Here is the CSM execution flowchart:

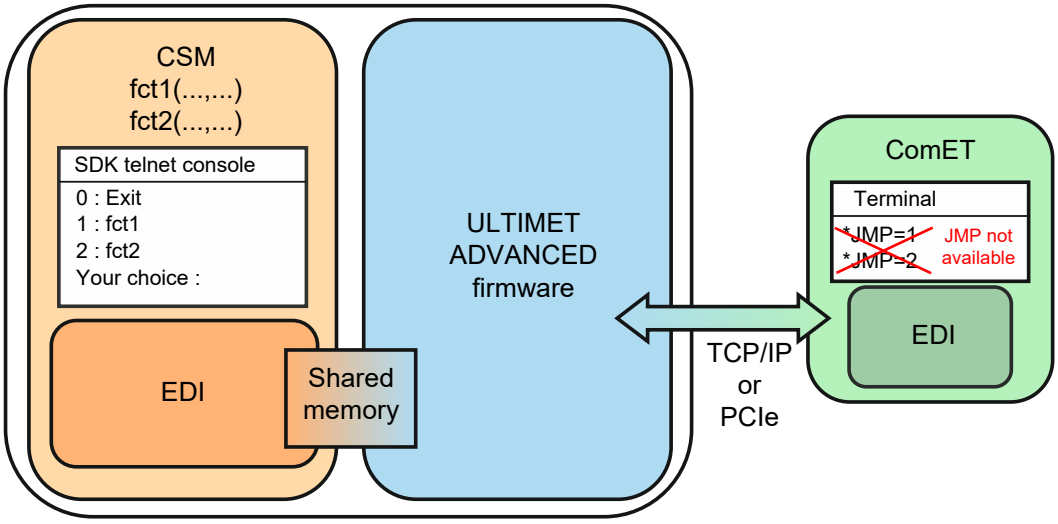


As explained in §20.1.1, this flowchart shows the 2 modes in which the CSM can work:

1. On the left side, we can see the “production” mode, where the CSM functions are executed by *JMP commands sent from a host application (e.g. ComET). In this mode, a UA-to-CSM command server is started. This server waits for *JMP and *END commands sent by host application and relayed by UA-firmware. In this mode, no user input is required. The CSM must be compiled with `REMOTE_CONTROLLED` pre-processor definition.



2. On the right side, we can see the “development” mode. In this mode, a menu displaying all available CSM functions is shown on the SDK telnet console (refer to §23.1.2 to create such a console). The user can enter the function number he wants to execute. Of course, the UA-to-CSM command server is not started in this mode. To allow this mode, the CSM must be compiled **without** `REMOTE_CONTROLLED` pre-processor definition.



20.3 EDI update

As mentioned earlier, the CSM uses an EDI version compiled for FreeRTOS that is the interface with the ULTIMET ADVANCED firmware. The CSM template that is provided, as a .zip file, contains the latest EDI version that was available when the CSM template was released. The name of the CSM template zip file contains the EDI version that is included in it., e.g. `CSMTemplate-EDI4.27B-UA3.23B.zip`.

Once a CSM project has been started based on a given version of the CSM template, it may be necessary to update the EDI that is included in the project. The EDI files used by a CSM project based on the template are in the directory `<CSM Project directory>\DevEnv\FreeRTOS\edi`.

EDI versions are also provided as zip files and there is one for FreeRTOS named `EDI-<version>-freertos-c.zip`.

To update the EDI used by a CSM project:

- Go to `<CSM Project directory>\DevEnv\FreeRTOS\edi`
- Copy `EDI-<version>-freertos-c.zip` into this directory
- Unzip here the EDI file; this will create a directory called `EDI-<version>`
- Remove the old `inc` and `lib` directories that were in `<CSM Project directory>\DevEnv\FreeRTOS\edi`
- Copy or move the files and directories in `EDI-<version>` directly to `<CSM Project directory>\DevEnv\FreeRTOS\edi`
- Rebuild the project

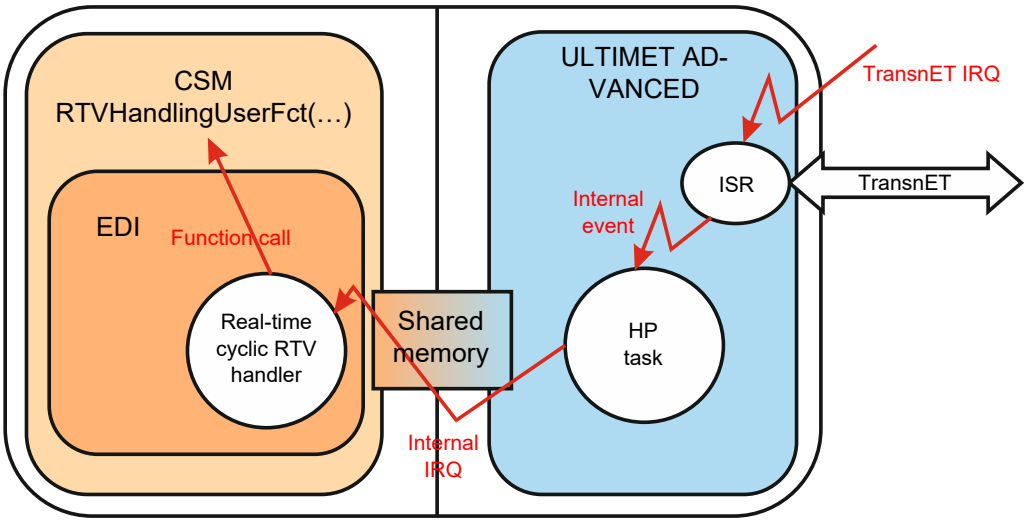
A CSM project can also be compiled on Windows with VisualStudio. The procedure is exactly the same with the CSM’s template directory being `<CSM Project directory>\DevEnv\VisualStudio\edi` and the EDI update package named `EDI-<version>-w32-vs2019-c.zip` or `EDI-<version>-w64-vs2019-c.zip`.

21 CSM special features

Additional CSM features are available for application development. They are not included in the basic CSM template provided by ETEL.

21.1 High priority real-time RTV handler

With this feature, a CSM can implement a real-time user function that is cyclically called by the ULTIMET ADVANCED firmware. The execution cycle of this user function is set with a user-defined multiple of the TransnET interrupt period (100 µs, 200 µs, 400 µs, etc.). The real-time RTV handler is very responsive, because it is executed with very high priority.

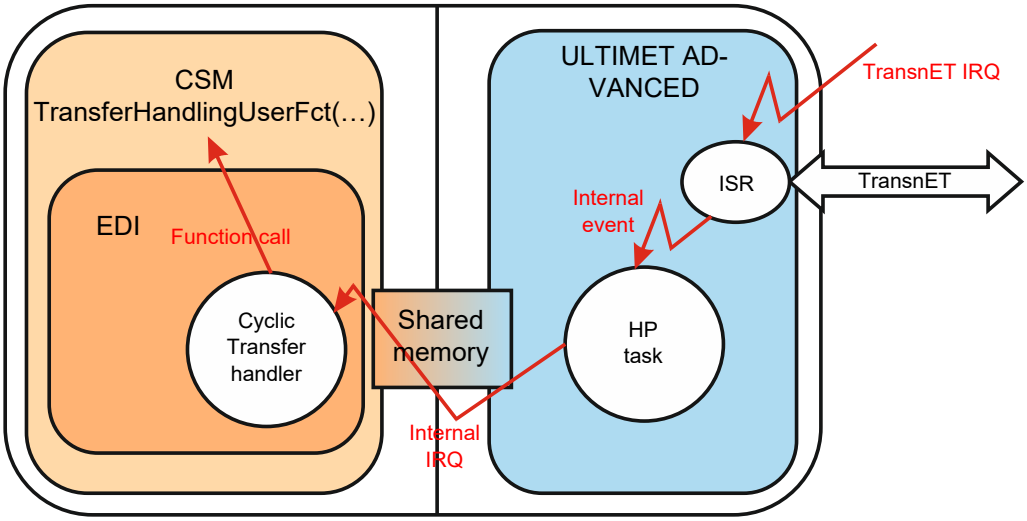


A use case example of this feature is the control of a system using a secondary position measurement system:

- Measurements of the secondary position measurement system are published on TransnET via RTV.
- The ULTIMET ADVANCED firmware triggers the CSM to execute the user function at a defined frequency.
- The user function retrieves the measurements from the RTV, computes the system axes position using a specific transformation matrix and sends new target positions via another RTV.
- The axes are configured to get their target position from these RTV.

21.2 Medium priority real-time Transfer handler

A second type of real-time user function is available. The principle is similar to the high priority RTV handler above, but the real-time Transfer handler is executed with a medium priority level. It is thus less responsive, because the Transfer handler will be “served” after the RTV handler.



A use case example of this feature is the implementation of a real-time camera auto-focus functionality:

- The CSM configures the ULTIMET ADVANCED SPI interface to transfer sensor data from a third-party distance measurement system, measuring the Z focus distance of the camera.
- The ULTIMET ADVANCED firmware triggers the CSM to execute the user function at a defined frequency.
- The user function retrieves the measurements from the data buffer filled by the SPI interface, computes the new Z target position and sends it via a TransnET RTV.
- The AccurET controlling the Z axis is configured to get its target position from this RTV.

21.3 Getting the current UA UTC time from a CSM

The time on the ULTIMET ADVANCED is set as described in §8.6. This time can be read from the CSM by calling the "time(time_t)" function:

```
#include <time.h>
time_t myTimeUTC;
time(&myTimeUTC);

// To print the time:
printf("Now is %s", ctime(&myTimeUTC));
```

22 CSM control from a host application

A host application (e.g. ComET) can control the execution (start and stop) of a CSM. Moreover, a host application can also trigger the execution of CSM high-level functions, but only if the CSM has been properly configured for enabling this functionality (refer to §20.1.1 for a detailed description on how to program the CSM to start the ULTIMET ADVANCED CSM command interface server).

22.1 Start a CSM

A host application can start a CSM using the **STARTCSM** command.

Command	***STARTCSM = <P1>**	
Description	Starts a customer software module	
Parameter	**Description**	
<P1>	CSM number, always 1	

ULTIMET ADVANCED will search for the requested CSM file named `CSM1.bin` in the CSM reserved folder in the micro-SD card (refer to §22.5 for a detailed description on how to store CSM on the micro-SD card). Starting a CSM takes approximately 10 to 15 seconds. This command does not wait for the CSM to be started. Refer to §22.2 for a proper CSM start.

Remark: Only one CSM can be started at a time. If a STARTCSM command is processed while a CSM is already running, the latter is abruptly stopped and the new one is started.

It is possible to have a CSM start automatically when the ULTIMET ADVANCED boots by setting the parameter ***K16**.

K	Name	Comment
***K16**	CSM autostart	Depth 0: It has the same meaning as the *STARTCSM command parameter. The ULTIMET ADVANCED will automatically start CSM1.bin when set to 1. Other values than 1 are not valid. Depth 1: CSM high level function to start. Depth 2: Thread in which to start function.

22.2 Waiting for a CSM to be started

Before any interaction with a CSM, the user must wait for the CSM to be correctly started. Monitoring register ***M100** is a bit field giving an indication if the CSM is running or not:

- Bit#0 indicates if the CSM is running when the CSM interface server is started (REMOTE_CONTROLLED).
- Bit#1 indicates there is an active connection between the CSM and the UA (resulting from an EDI dsa_open_u call).
- Bit#2 indicates if the CSM core#3 is running or in idle state.

M	Name	Comment
***M100**	CSM activity	CSM activity status: bit#0: CSM activity in REMOTE_CONTROLLED mode: 1 running, 0 not running; bit#1: CSM-UA connection state: 1: established, 0: none; bit#2: CSM core#3 status: 1: running, 0: idle state

The **WAITCSM** command can be used to wait until the CSM is running.

Command	***WAITCSM = <P1>**		
Description	Wait for the customer software module to start. Wait for the customer software module to start. The parameter defines which bit (only one bit) of *M100 is taken into account. (Bit# 0 implies bits# 1 and 2, and bit#1 implies bit#2.)		
Parameter	**Description**	**Value**	**Description**
<P1>	Defines the type of CSM wait	1 (bit#0) 2 (bit#1) 4 (bit#2)	CSM running in REMOTE_CONTROLLED mode (standard case) CSM connection established CSM core running

22.3 Run a CSM high-level function

If the CSM has been programmed to start the ULTIMET ADVANCED CSM command interface server, a host application can trigger the execution of a CSM high-level function using the **JMP** command.

Command	***JMP[.<AXIS] = <P1>, [<P2>]**
Description	Start CSM high level function <P1> in thread number <P2>
Parameter	**Description**
<P1>	CSM high level function number
<P2>	Thread number. If <P2> is not mentioned or equal to zero, the default thread number 1 is executed. The allowed range of thread numbers is from 1 to 8.”

Remark: This command can only be used if the CSM is running. The CSM might have been started through the Xilinx SDK environment or via ComET using the *STARTCSM command.
The CSM must have been compiled using the `REMOTE_CONTROLLED` symbol defined.

The monitoring register ***M97** indicates what threads are running. Refer to §20.1.1 for more information.

It is possible to see in *M63 (refer to §8) if some of the threads (as specified by<P2> of the *JMP commands) are running or not. Parameter ***K297** is the mask of the thread whose running status is reflected in bit#31 of *M63.

K	Name	Comment
***K297**	Running thread mask	If one of the threads in this mask is running, bit#31 of *M63 will be set to 1. If bit#31 of *M63 = 0, none of the threads specified by *K297 are running (other threads may)

22.4 Stop a CSM

If the CSM has been programmed to start the ULTIMET ADVANCED CSM command interface server, a host application can stop the execution of a CSM high-level endless function using the **END** command.

Command	***END[.<AXIS>]=<P1>**
Description	Stops the CSM thread defined by <P1> or stops all threads if command without <P1>.
Parameter	**Description**
<P1>	Thread to stop

Remark: To stop a CSM high-level endless function, it must have been programmed as “interruptible” (refer to §20.1.1 for an example on how to program a function as “interruptible”).

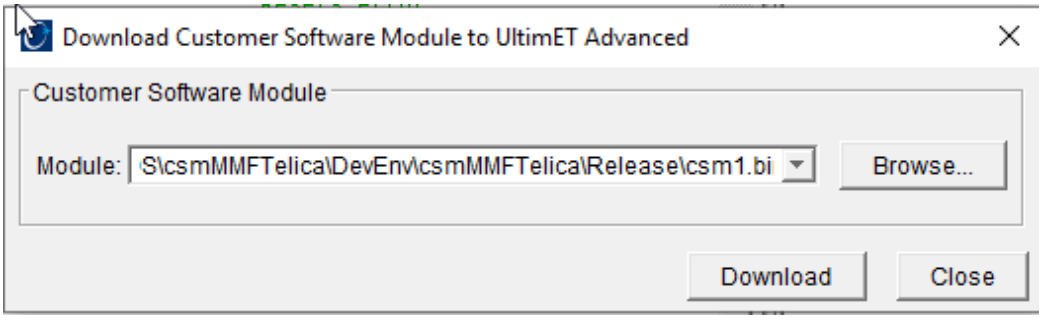
Commands *HLO, *HLT, and *HLB may stop those threads. Refer to §15.4.4.4 for more information.

To completely stop the CSM use the command **STOPCSM**. This will not only stop any running threads, but also the entire execution of the application itself.

Command	***STOPCSM**
Description	Stops a customer software module.

22.5 Store a CSM on the permanent storage

ComET can be used to download a CSM to the ULTIMET ADVANCED permanent storage (micro-SD card) by clicking on **’Controller’** in the menu bar and then on **’Download Customer Software Module’**. The following dialog box is displayed:



Select the binary file you have generated (refer to §19.2 for more details on how to create a CSM). The CSM can also be downloaded using host application with the following EDI function:

```
int dsa_download_customer_software_module_s(DSA_MASTER *master,
                                            const char* csm_file_name,
                                            unsigned int csm_no, dword flag,
                                            long timeout,
                                            DSA_PROGRESS progress_fct,
                                            void *puser);
```

Finally, from ULTIMET ADVANCED firmware version 3.19A, the ComET tool System Configuration can backup and restore the downloaded CSM, as well as all user data present in sub-directory Config of user SD.

Caution: If you download a new CSM, the old one will be overwritten.

23 CSM troubleshooting

23.1 Debug stdio/stdout

In a development environment, the Xilinx SDK console is the standard input/output stream (stdin/stdout) mechanism of a CSM with the connection established via the JTAG interface. This mechanism has two main disadvantages:

- The message output is slow, approximately 3ms per each `printf` call.
- If the JTAG cable is not plugged to the ULTIMET ADVANCED (connector X106), a call to the stdout `printf` function will make the CSM hang indefinitely. Therefore, calls to `printf` function in the CSM are not acceptable for the RELEASE version of the CSM. It is mandatory to link the RELEASE version of CSM with the specific FREERTOS_SOFTWARE_PLATFORM which has the stdin/stdout disabled, so that the “printf” call will not display anything but your application will not hang.

On the other hand, it is fundamental to have an effective way of inspecting the execution of a CSM for troubleshooting issues observed on already released products.

23.1.1 TCP/IP debug server

The ULTIMET ADVANCED includes a third TCP/IP connection (connector X4) dedicated to the CSM. Parameter ***K708** is used for configuring this connection as follows:

K	Comment
***K708:0**	IP address of the CSM TCP/IP server
***K708:1**	Subnet mask of the TCP/IP configuration of the CSM
***K708:2**	IP address of the gateway for the TCP/IP configuration of the CSM

The MAC address used for this connection is set at production time and cannot be changed. It can be read back by getting *M707:

M	Name	Comment
***M707**	Ethernet X4 MAC address	MAC address set for connector X4

Remark: The CSM TCPIP server cannot be connected to a 10-100 MB network or switch.

The EDI API provides an alternate input/output stream mechanism to be used with the CSM based on a TCP/IP interface. With this mechanism, the user can interact with the CSM through a Telnet session.

The EDI-DSA library provides the following interfacing functions:

Function	Description
dsa_csm_dbg_server_start	Initialize the TCP/IP based debug server
dsa_csm_dbg_connected	Check if a user is connected to the debug server
dsa_csm_dbg_wait_connected	Wait until a user is connected to the debug server
dsa_csm_dbg_printf	Display a character string (same use as `printf` C function) on the user side
dsa_csm_dbg_scanf	Request a user input (same use as `scanf` C function)
dsa_csm_dbg_server_stop	Stop the TCP/IP based debug server
dsa_csm_dbg_flush	Flush the rolling buffer and displays the stored logs on telnet if connected)
dsa_csm_dbg_sflush	Flush the rolling buffer after having copied the stored logs in the pre-allocated buffer passed as argument

To use this mechanism, the CSM has to initialize the debug server by calling the function `dsa_csm_dbg_server_start()` with three arguments:

- `master` ULTIMET ADVANCED device object from which the CSM TCP/IP settings can be obtained from register *K708 in order to configure the CSM debug server.

- `telnet_port` to connect to the CSM debug server, the user must provide the corresponding port. This port number is configurable.

- `buffer_size` the size of an optional circular buffer. If different than 0, the output is stored in the buffer instead of being displayed on Telnet server.

The two typical use cases of the CSM debug server are:

- Output information to the user and/or developer about the execution status of the CSM.
  In this use case, the developer needs to include this code in the CSM to initialize the debug server:

```
// From here, the user input/output are done on telnet server
dsa_csm_dbg_server_start(ui_fct_param.my_system->csmDevices[ULTIMET_IDX].dev,
                         DEBUG_PORT, 0);
```

Then, use function `dsa_csm_dbg_printf` to display information as required:

```
dsa_csm_dbg_printf("CSM started\n");
...
dsa_csm_dbg_printf("RTV slots assigned\n");
...
```

As soon as the user/developer initiates a Telnet session with the CSM on the `DEBUG_PORT`, the messages will be displayed on the Telnet terminal.

Finally, when the log functionality is no longer required, the CSM can close the debug server by calling function `dsa_csm_dbg_server_stop()`.

- Display a “menu” for the user/developer to select an option and wait for his input for selecting the corresponding execution flow (only makes sense during the development phase). In this case, after starting the debug server using function `dsa_csm_dbg_server_start()`, it is important to wait for the user/developer to create the `Telnet` connection using function `dsa_csm_dbg_wait_connected()`. Otherwise, it can happen that the CSM sends the menu prior to the establishment of the Telnet connection, in which case the user will clearly miss it and not know what to do next.

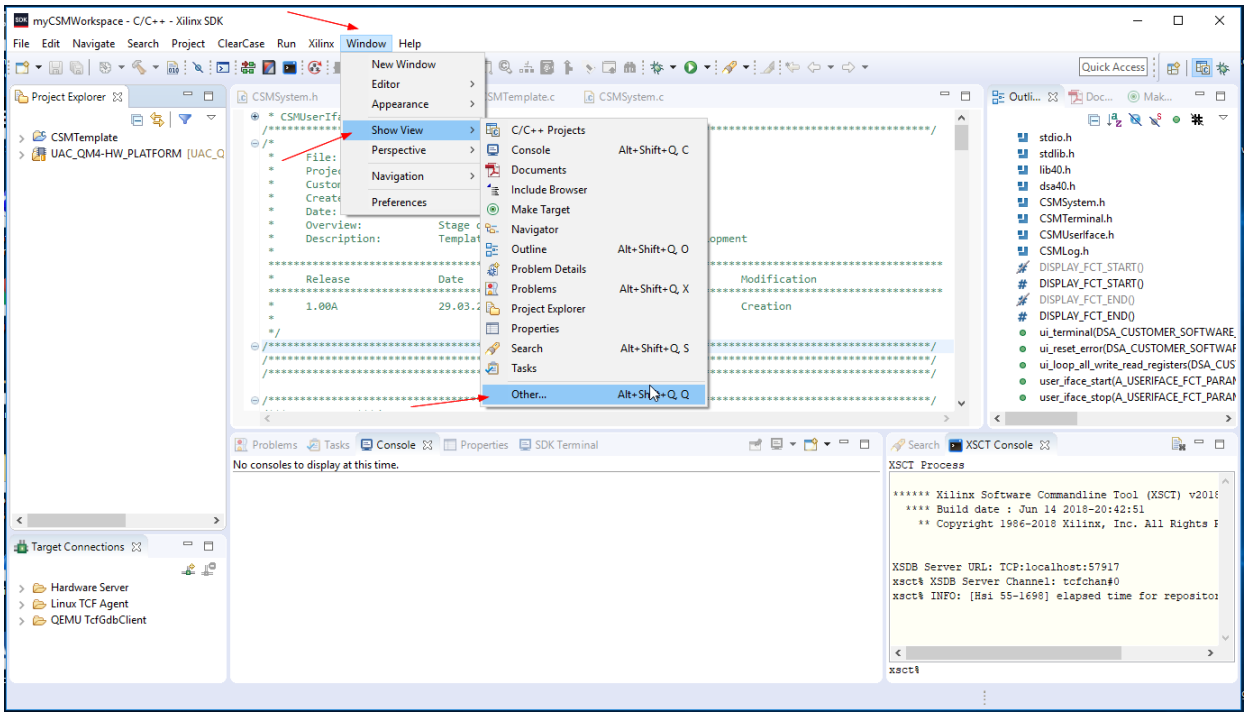
```
//From here, the user input/output are done on telnet server
dsa_csm_dbg_server_start(ui_fct_param.my_system->csmDevices[ULTIMET_IDX].dev,
                         DEBUG_PORT, 0);
#ifndef REMOTE_CONTROLLED
    dsa_csm_dbg_wait_connected();
#endif

dsa_csm_dbg_printf("Start user Interface\n");
user_iface_start(&ui_fct_param);
```

Remark: CSMs requiring user input are not recommended for production releases.

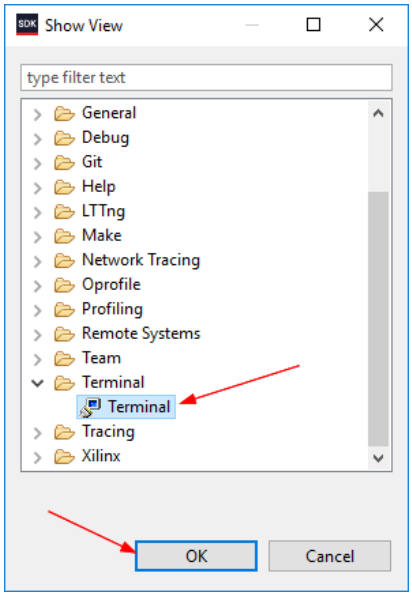
Caution: Displaying logs is time-consuming. Logs are useful during development but should be minimized for production release, even when displayed using Telnet. As Telnet is based on TCP/IP, it induces a synchronization between it and the CSM, which will considerably slow down the CSM.

23.1.2 Opening a Telnet terminal in the Xilinx SDK

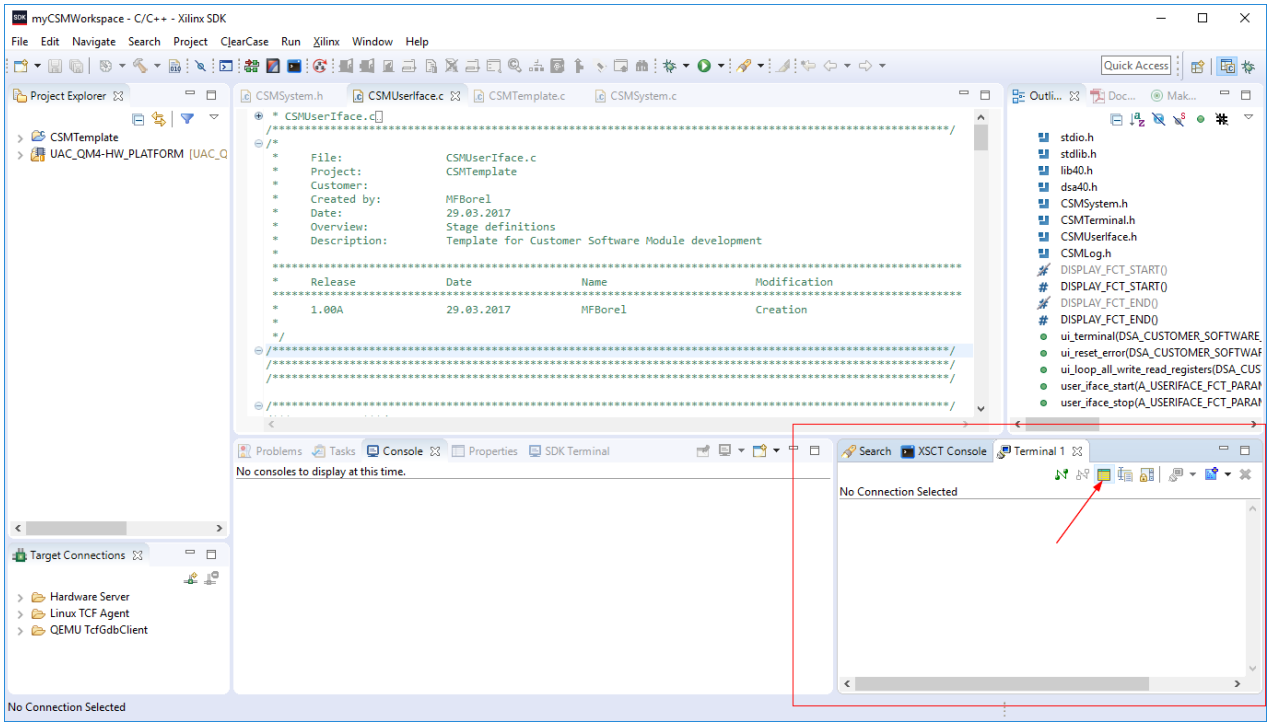
Select menu “Window”, “Show View” and “Other…”:



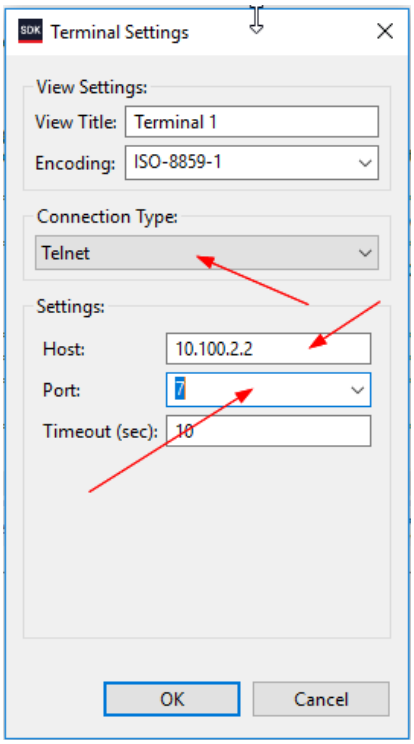
Select “Terminal” view and press “OK”:



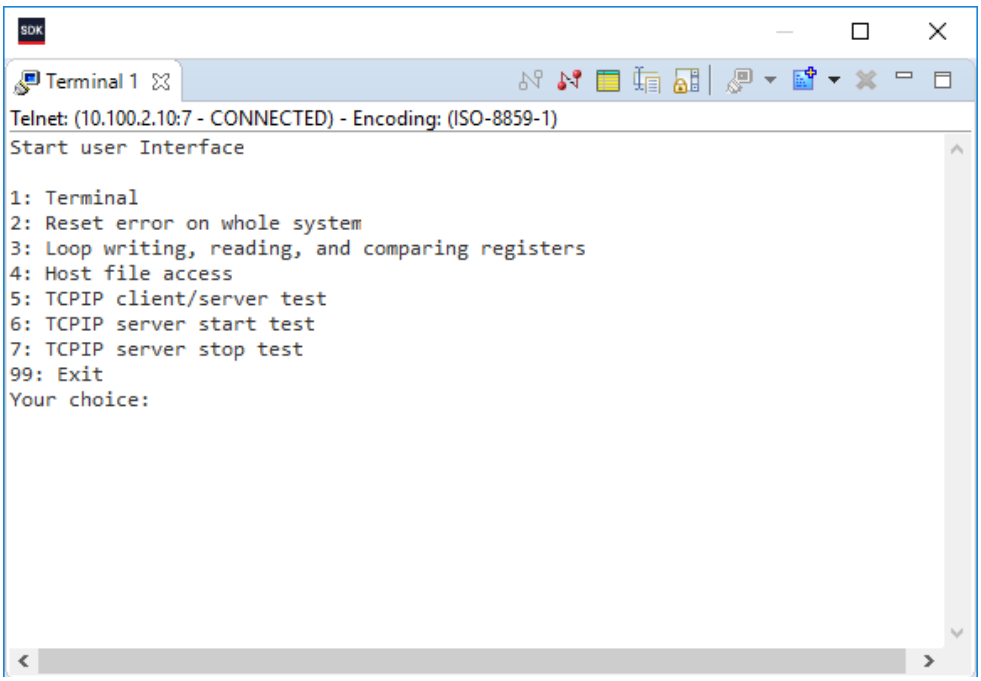
The terminal will be displayed in the SDK IDE. Select the setting icon to modify the terminal settings:



Change the Connection type to Telnet, provide the CSM IP address and Port of the CSM debug server (the CSM IP address can be found in register *K708:0 and the CSM debug port is the one provided when calling function `dsa_csm_dbg_server_start`) and press OK.



A Telnet connection to CSM can only be initiated once the CSM debug server has started.



Caution: Do not disconnect the Telnet session while the CSM is hanging on a `dsa_csm_dbg_scanf`, otherwise the CSM will be eternally waiting for the scanf input and will hang there.

23.2 File logging

The standard SD card is available exclusively for the user's embedded application data (i.e. CSM). The developer of a CSM can store its own data, including data for troubleshooting purposes.

The EDI API provides the necessary functions for file management on this SD card:

Function	Description

`int dsa_host_file_open(`
`DSA_DEVICE *dev,`
`const char* filename,`
`char *open_mode,`
`dword flag,`
`DSA_HOST_FILE**filep);`
	Creating or opening a file
	`<filename>` is the complete file path.
	The folder must have been created.
	The root folder is specified using «/».

`int dsa_host_file_write(`
`DSA_DEVICE *dev,`
`DSA_HOST_FILE *filep,`
`const void *buffer,`
`size_t buffer_size,`
`size_t *written_size);`
 	
`int dsa_host_file_printf(`
`DSA_DEVICE *dev,`
`DSA_HOST_FILE *filep,`
`const char *format,...);`
 	
`int dsa_host_file_vprintf(`
`DSA_DEVICE *dev,`
`DSA_HOST_FILE *filep,`
`const char *format,`
`va_list vargs);`
	Writing to a file

Function	Description

`int dsa_host_file_read(`
`DSA_DEVICE *dev,`
`DSA_HOST_FILE *filep,`
`void *buffer,`
`size_t buffer_size,`
`size_t *read_size,`
`DSA_HOST_FILE *filep);`
 	
`int dsa_host_file_gets(`
`DSA_DEVICE *dev,`
`DSA_HOST_FILE *filep,`
`char* buffer,`
`size_t buffer_size);`
	Reading from a file

`int dsa_host_file_close(`
`DSA_DEVICE *dev,`
`DSA_HOST_FILE *filep);`
	Closing a file

`int dsa_host_file_eof(`
`DSA_DEVICE *dev,`
`DSA_HOST_FILE *filep,`
`ebool *end_of_file);`
	Verifying EOF

`int dsa_host_file_mkdir(`
`DSA_DEVICE *dev,`
`const char* dirname);`
	Creating a folder
	`<dirname>` is the complete folder path.
	The parent folder must exist.
	The root folder is specified using «/».

`int dsa_host_file_delete(`
`DSA_DEVICE *dev,`
`const char* filename);`
	Deleting a folder or a file
	`<filename>` is the complete file or folder path to be deleted.
	If a folder is specified, its content will be deleted recursively.
	`<filename>` syntax can contain wildcards with the following restriction:
	• Wildcard * can be used at the beginning and/or end of the name
	• Wildcard ? can be used without restriction, except before or after the wildcard *

Remark: Logs are useful at every stage of the product life-cycle. However, a good balance between logs and real-time execution is required.
It is recommended to use files for logging in production releases, since they are helpful for troubleshooting issues at customer site.

Examples:

Open a configuration file and read its content:

```
int read_config(A_SYSTEM *psystem)
{
    DSA_MASTER *mst = psystem->csmDevices[ULTIMET_IDX].dev; // Retrieve the
                                                            // underlaying EDI
                                                            // object
    DSA_HOST_FILE *f = NULL; // Host file pointer
    char buffer[256];
    ebool eof;
    int err, err1;

     // Open the file
    if ((err = dsa_host_file_open(mst, "Config/configFile.txt", "r", 0, &f))) {
        DSA_EXT_DIAG(err, NULL);
        goto _error;
    }

    // Reads all lines until End Of File is reached
    for (;;) {
        if ((err = dsa_host_file_gets(mst, f, buffer, sizeof(buffer)))) {
            if ((err1 = dsa_host_file_eof(mst, f, &eof))) {
                DSA_EXT_DIAG(err1, mst);
                goto _error;
```

```
                }
                if (eof)
                    break;
                //Process the config file line
                ...
            }
        }

_error :
    // Close the file
    if (f) {
        dsa_host_file_close(mst, f);
        f = NULL;
    }
    if (err) return err ;
    if (err1) return err1 ;
    return 0 ;
}
```

Log on screen and in a file:

```
// This example displays the log entries on the screen and also stores them in a
// file.
// In this case, the file is opened and closed each time the function is called.
// This has the advantage that the file is always up-to-date, even if the
// application crashes, but the disadvantage is that it takes more time.
// Another way would be to open the file once and, in this function, just put
// the log in the file.
// But you must be aware that the logs are really stored in the file when the
// file is closed, so it is advised to close regularly the file.

int myLog(A_SYSTEM *psystem, char *format, ...)
{
    DSA_MASTER *mst = psystem->csmDevices[ULTIMET_IDX].dev; // Retrieves the
                                                            // underlaying EDI
                                                            // object
    DSA_HOST_FILE *f = NULL; // Host file pointer
    int err = 0;             // The error returned by function call
    va_list vargs;           // The variable arguments

    // Open the file
    if ((err = dsa_host_file_open(mst, "LogFile.txt", "a+",
                                  DSA_HOST_FILE_AUTOFLUSH_FLAG, &f))) {
        DSA_EXT_DIAG(err, NULL);
        goto _error;
    }

    // Retrieves the list of arguments
    va_start(vargs, format);

    // Print the log on telnet console
    dsa_csm_dbg_vprintf(format, vargs) ;

    // Store the log in the file
    if (err = dsa_host_file_vprintf(mst, f, format, vargs)) {
        DSA_EXT_DIAG(err, NULL);
        goto _error;
    }

    // Error management
_error :
   // Close the file
    if (f) {
        dsa_host_file_close(mst, f);
        f = NULL;
    }
    return err ;
}
```

Logs can be accessed via an FTP connection. The procedure is described in the next chapter.

23.3 FTP access to User SD card

An FTP server is present in the ULTIMET ADVANCED and allows the user to access the user SD card files. Connection login and passwords are:

- login: guest
- password: guest

Two ways to do an FTP access are provided here:

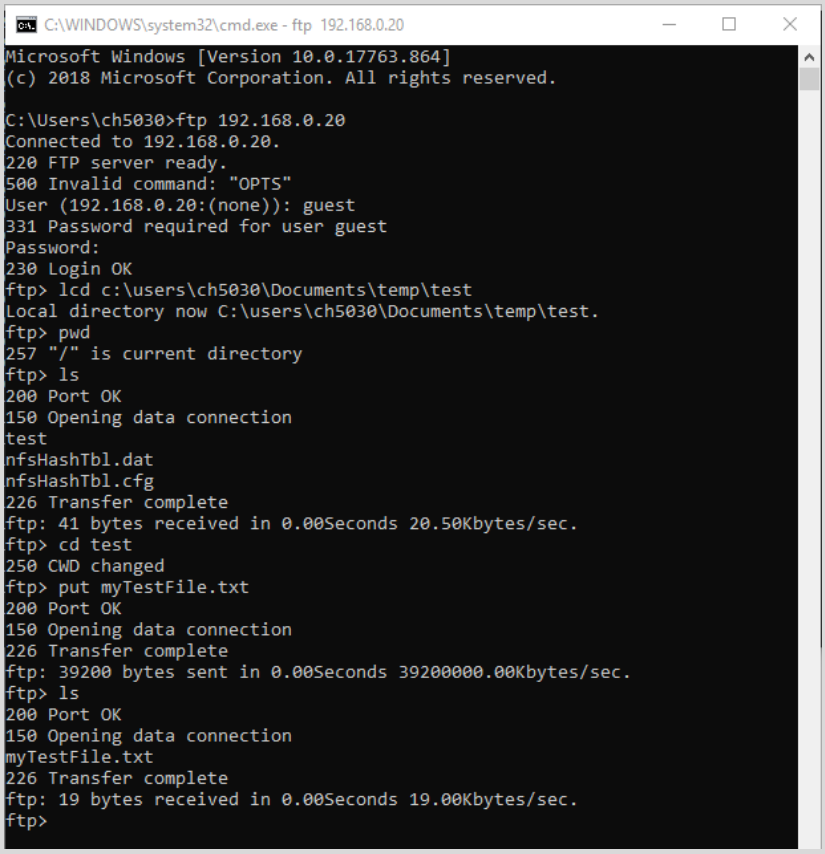
- Windows command shell.
- Windows explorer.

23.3.1 FTP from windows command shell

Windows Command shell example

This example does:

- An ftp connection to 192.168.0.20, login and password.
- A local (PC) directory change to a «test» folder containing a file «myTestFile.txt».
- Command “pwd”: shows current remote path: « / », which is the root path on user SD card.
- Command “ls”: shows SD card content.
- Command “cd test”: remote change directory to «test» folder.
- Command “put myTestFile.txt: copy «myTestFile.txt» from local test folder to remote test folder.
- Command “ls”: shows SD card «test» folder content.

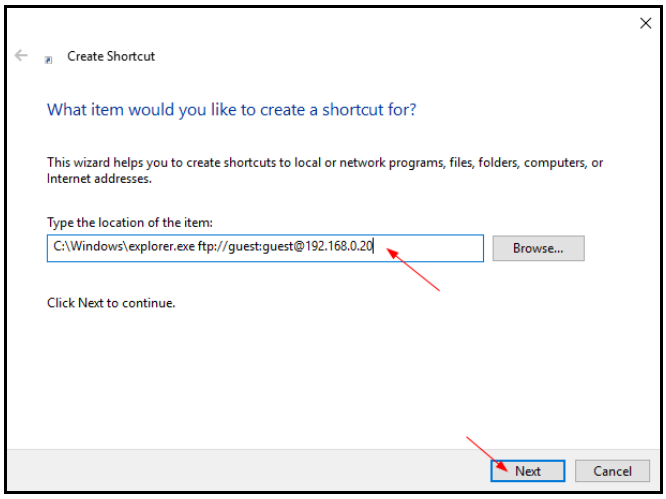


Remark: The error message "500 Invalid command: OPTS", in the picture above, is due to Windows OS. It has no effect on the FTP use.

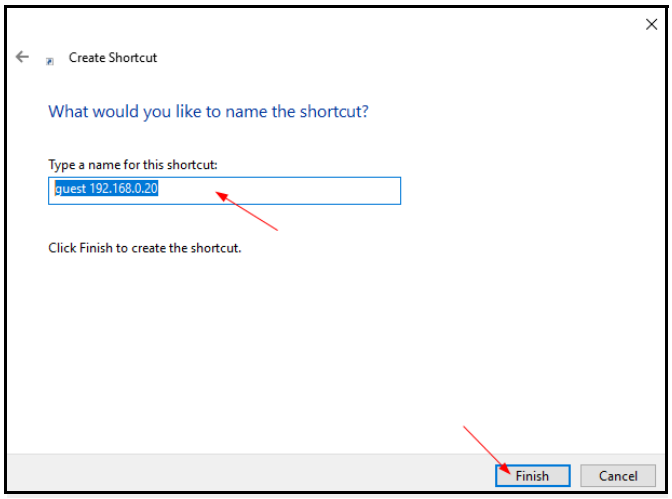
23.3.2 FTP Access through Windows Explorer

This chapter shows the process to access the User SD card in the Windows Explorer:

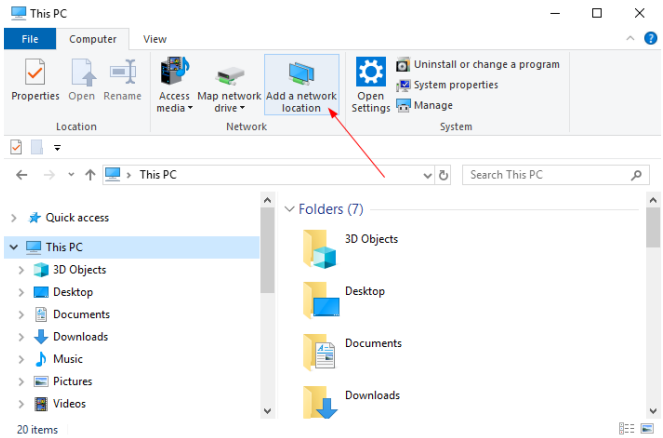
- Option 1 - Create a Windows Explorer shortcut:
    - C:\Windows\explorer.exe ftp://guest:guest@<UA_IP_address>



- Example: C:\Windows\explorer.exe ftp://guest:guest@192.168.0.20

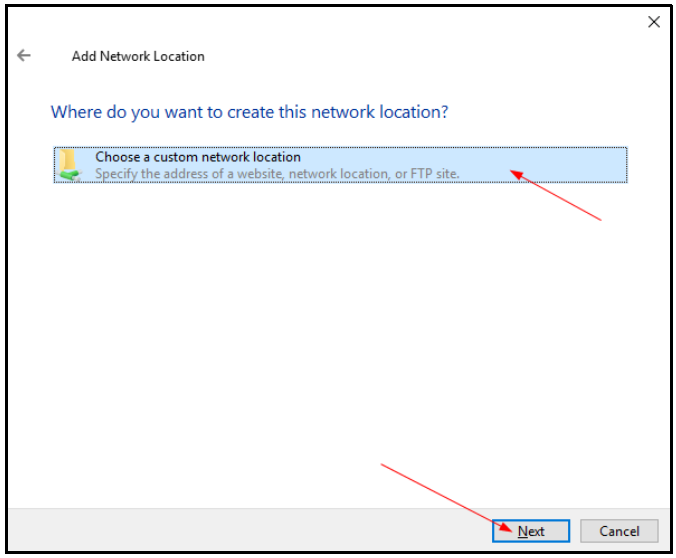


- Option 2 - Add a network location:
  - Click on “Add a network location”.

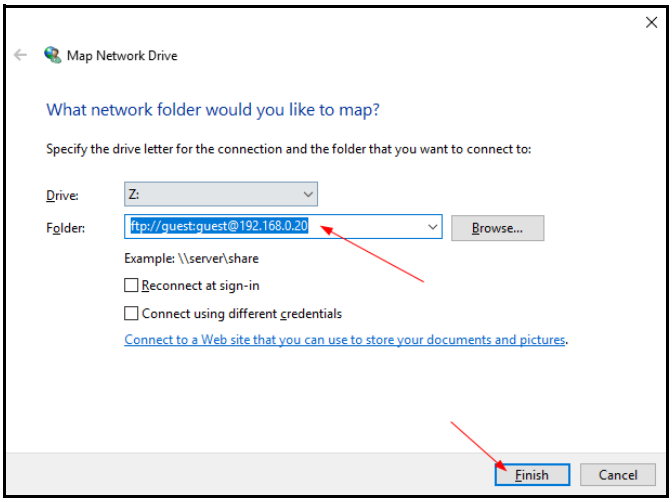


  - Click next in the welcome panel.

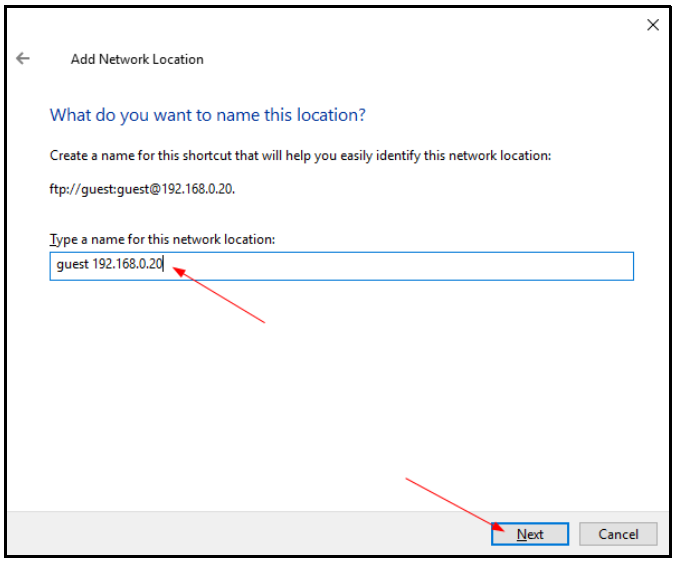
- Click on “Choose a custom network location” and then on “Next”.



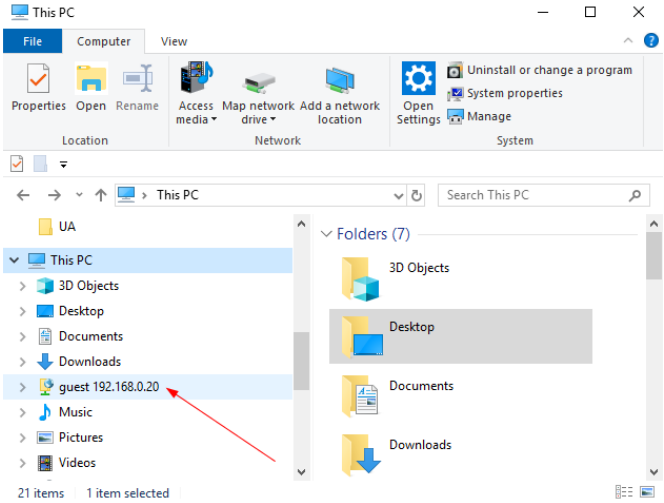
- Enter the following text: ftp://guest:guest@<UA_IP_address>.
  - Here: ftp://guest:guest@192.168.0.20



- Click next and name your connection (e.g. “guest 192.168.0.20”).



- Then your connection will be visible in the windows explorer.



24 EDI functions not supported in a CSM

1. EDI C++ interface is not supported.

2. The following EDI functions cannot be used in a CSM:

- dsa_debug_sequence…
- dsa_start/stop_customer_software_module_s/a
- dsa_start/stop_customer_software_module_function_s/a
- dsa_system_configuration…
- dsa_device_configuration…
- etb_etcom_download_firmware(_ex)
- etb_get_firmware_info
- Generically all `etne` functions

25 Special note about dynamic memory allocation

FreeRTOS has been configured to use the most powerful heap management, minimizing the memory fragmentation. This needs the use of the EDI layer to manage dynamic allocation. This means that the programmer must avoid using the standard C dynamic memory functions and use instead the following EDI macros:

```
// Allocates a chunk of memory for <nbItem> of <itemSize> bytes, and initialize
// the chunk with 0
void *p = CALLOC(size_t nbItem, size_t itemSize):

// Allocates a chunk of memory of <size> bytes
void *p = MALLOC(size_t size):

// Free the chunk of memory pointed by <p>
FREE(void* p):

// Reallocate the chunk of memory pointed by <oldP> to a new chunk of memory of
// size <newSize>
void *p = REALLOC(void *oldP, size_t newSize)
```

In case the user accidentally uses standard malloc, calloc or free C functions, it is recommended to use the wrapped functions provided by EDI. The CSM template defines a WRAPPING build variable containing this wrapping instruction. Please refer to the next chapter to see the WRAPPING definition, also used for `printf` fonction.

26 Special note about printf suite functions

`Printf` functions from FREERTOS use dynamic allocation to display floating point values. To avoid that, ETEL EDI uses a third-party library implementing the `printf` suite functions without dynamic allocation.

To use them, a CSM application must specify the link option -wrap (see example below). This allows to continue using the default `printf` functions in the CSM code. The linker -wrap option will replace the standard `printf` function with the EDI's one.

The CSM template project defines a build variable WRAPPING, which defines the complete link option which must be used.

```
WRAPPING = -Xlinker --wrap -Xlinker vsnprintf -Xlinker --wrap -Xlinker snprintf -
Xlinker --wrap -Xlinker sprintf -Xlinker --wrap -Xlinker vprintf -Xlinker --wrap
-Xlinker printf -Xlinker --wrap -Xlinker malloc -Xlinker --wrap -Xlinker calloc -
Xlinker --wrap -Xlinker free
```

You can notice above that a wrapping is also defined for standard libc malloc, calloc, free, allowing to call the FREERTOS specific heap management (heap_4.c) used by EDI.

THIS PAGE IS INTENTIONNALLY LEFT BLANK

Chapter E: Appendixes

27 ULTIMET ADVANCED – UltimET Light differences

ULTIMET ADVANCED remains compatible with UltimET Light features. There are however some differences that must be highlighted. These differences are grouped in three paragraphs:
- New or improved features.
- Features that behave differently.
- Features that are not or not yet in ULTIMET ADVANCED.

27.1 New features

27.1.1 Customer Software Module (CSM)

The CSM is the evolution of UltimET Light “compiled sequences”. The CSM is an embedded EDI application. Refer to chapter D (page 149) for more information.

27.1.2 Several application interfaces (TCP/IP – PCIe – CSM)

ULTIMET ADVANCED allows the customer to be connected on it through several application interfaces at the same time:
- TCP/IP (2 physical connections: X1 – X2).
    - Port 1129 allowing two sockets opened in parallel.
- PCIe connection linked to PC via an extension cable.
- CSM: real-time embedded application.

UltimET Light application interface is one of PCI, PCIe or TCP/IP.

27.1.3 Configurable IOs

On ULTIMET ADVANCED there are 8 configurable IOs that can be configured as:
- Input,
- Output,
- or for SPI (Serial Peripheral Interface) communication.

On UltimET Light there are 4 Inputs / 4 Outputs (not configurable) in the PCI – PCIe versions, and 2 Inputs / 2 Outputs on the TCP/IP version.

Refer to chapter C (page 33) for more information.

27.1.4 SPI (Serial Peripheral Interface) feature

This feature is new and does not exist on UltimET Light. Refer to chapter C (page 33) for more information.

27.1.5 More RTV slots accessible via ASR/ASW

More Real-Time Values slots can be accessed or shared by ULTIMET ADVANCED.

	Read from TransnET (ASR)	Write on TransnET (ASW)
ULTIMET ADVANCED	64	32
UltimET Light	16	8

The range of ASR monitoring register is new:
- M/MF from 310 to 373.
- ML/MD from 310 to 341.

27.1.6 SD cards

SD cards are present on ULTIMET ADVANCED:
- micro-SD card for ETEL internal data storage (e.g. internal logging) and CSM binary files.
- user SD card for user CSM related data storage.

These SD cards offer extended data storage possibilities, compared to UltimET Light (flash storage for firmware, registers and compiled sequences).

27.2 Features that behave differently

27.2.1 PCIe watchdog

The PCIe watchdog functionality's goal is to raise an error if RTV cyclic tasks or cyclic tasks (SPI cyclic handling) are not executed at the right moment.

ULTIMET ADVANCED raises an error if the latency time between a RTV PCIe/CSM interrupt and the handler completion is longer than a defined limit (programmed through EDI).

UltimET Light PCI/PCIe watchdog raises an error if the latency time between a RTV cyclic interrupt and its acknowledgment is bigger than a defined limit (programmed through EDI).

27.2.2 Emergency halt features

In UltimET Light, *HLx commands behave as follows:
- As normal commands:
    - It aborts all pending commands.
    - It stops interpolation movements and disables active interpolated groups.
    - It stops sequences and waits for the sequence threads to finish (if not masked by *K144).
    - It waits for the interpolated groups to be disabled.
- As urgent commands:
    - It stops sequences but DOES NOT wait that the threads are finished (if not masked by *K144).
    - It aborts all pending commands.
    - It stops interpolation movements and disables active interpolated groups.

In ULTIMET ADVANCED, *HLx commands behave as follows:
- As normal commands:
    - It aborts all pending commands.
    - It stops interpolation movements and disables active interpolated groups.
    - It aborts any SPI cyclic task.
    - It stops CSM threads and waits for the threads' termination (refer to §15.4.4.4 for more information).
    - It waits for interpolated groups to be disabled.

- As urgent commands:
    - It aborts all pending commands.
    - It stops interpolation movements and disables active interpolated groups.
    - It aborts any SPI cyclic task.
    - It stops CSM threads and waits for threads' termination (refer to §15.4.4.4 for more information).

27.3 Features not in ULTIMET ADVANCED

27.3.1 Interpolation Unified Linear Move (ULM) mode

This interpolation mode is currently not implemented on ULTIMET ADVANCED.

27.3.2 Get / Free Gate command

Commands GGA and FGA are not implemented on ULTIMET ADVANCED.

27.3.3 Nodes status registers

The monitoring register *M520 that holds a copy of all the nodes status registers M63 is currently not implemented.

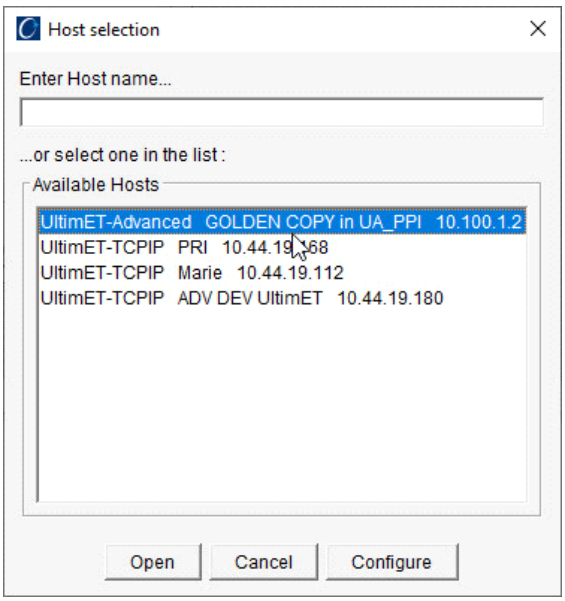
28 Troubleshooting

28.1 Recovery firmware

The ULTIMET ADVANCED sports a backup firmware called "Recovery firmware". The Recovery firmware is made up of all the boot components (boot loaders), a minimal operating system and a reduced features firmware. It is stored in flash and started automatically if anything wrong is detected at startup, like an incorrectly terminated firmware download, corrupt firmware files, micro SD card access issues. It is meant only to be able to download again a new firmware on the card. Thus it cannot run any CSMs nor write to any log files e.g.

There are several ways to identify that the firmware running is the Recovery firmware:

- The ULTIMET ADVANCED will be in error with code 4201. This error cannot be reset. It will disappear after a subsequent successful firmware download and reboot.
- In the ComET host selection, the name of the ULTIMET ADVANCED will have been prepended with "Recovery firmware in"



- Monitoring register *M626 gives the reason why the ULTIMET ADVANCED has jumped into the Recovery firmware.

M	Value	Condition / Issue	Recovery
*M626	0x10xx	Corrupted micro-SD card	Contact your provider
	0x1300	Corrupted flash	Contact your provider
	0x14xx	Corrupted micro-SD card	Download firmware
	0x15xx	Corrupted file	Download firmware
	0x16xx	The previous firmware download failed	Download firmware again
	0x1710		
	0x1720		
	0x1730		

If the problem persists, contact your provider.

28.2 Troubleshooting hints

Troubleshooting hints linked to an error raised by the ULTIMET ADVANCED is available in the errors list (§33.). Additional hints are provided hereafter in case of communication problems.

28.2.1 JTAG connection not accessible

If the JTAG connection cannot be established:
- Check the JTAG cable is correctly connected and the ULTIMET ADVANCED is powered.
- Then, check the driver is properly installed (both in USB and serial ports).
- If it is still not working, re-install the JTAG driver (available in Xilinx SDK, “install_digilent.exe”).

Finally if everything seems correct and it is still not possible to open the JTAG connection, please contact your provider.

28.2.2 ULTIMET ADVANCED PCIe connection not accessible

As a reminder, it is mandatory to power the ULTIMET ADVANCED before the host PC is booted up. In addition, if the ETEL JTAG extension cable is not used, it is required to remove the JTAG cable from the ULTIMET ADVANCED before starting the PC. Otherwise, the host PC is not able to properly initialize the PCIe interface.
If this initial procedure is respected and it is still not possible to open the communication over PCIe, here are some advice:
- Check the PCIe extension cable is well connected.
- Check the ULTIMET ADVANCED is “alive” as follow.
    - One of the X5 connector’s LEDs (either the green or red one) is blinking.
    - It is possible to communicate to ULTIMET ADVANCED through TCP/IP.
- Check the state of the PCIe host cable adapter. E.g. for the PCIe x4 Gen2 Host Cable Adapter from One Stop Systems, the two external LEDs PWR and CBL and the two board LEDs CBL and EDGE must be on. Otherwise it means there is a problem with the adapter. Please refer to the adapter manufacturer in such a case.
- Check the ULTIMET ADVANCED and WinDriver drivers are properly installed and no error is visible on drivers’ icons:

If it is still not possible to connect to the ULTIMET ADVANCED through PCIe, execute the configuration update script presented in §3.2.2. Then try to open again the communication.

If the ULTIMET ADVANCED driver is in warning state, try to plug the host adapter board in a different slot (refer to §28.1). If this is still not working, please contact your provider.

28.2.3 CSM troubleshooting

Refer to chapter D (page 149) for more information.

28.2.4 SPI troubleshooting

28.2.4.1 The data read on SPI is not correct

If read data are not correct, please check that:
- Cabling between master and slave is correct. Refer to ULTIMET ADVANCED hardware manual for the correct IO pin-out.
- IOs are correctly configured (registers *K800 to *K803). Refer to SPI documentation (§16) for detailed information.

- The SPI configuration is the same on both SPI devices. For ULTIMET ADVANCED, information on SPI clock mode (CPOL and CPHA), CrcMode, MSB or LSB first are described in (§16.2.1) for detailed information.

If still not working, please contact your provider.

28.2.4.2 Handler error

In SPI cyclic mode (§16.5), a user handler is called each SPI cyclic iteration. This handler may encounter error situations and these might not be managed directly in the handler (handler periodicity may be too high).

The advice is to use an error variable (either global or contained in a user structure accessible through the handler user_param parameter). Other threads of the user application will handle this error.

29. Commands list

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
*ASR[.<AXIS>] = <P1>, <P2>, <P3>	231		Gives the slot offset in the TransnET cyclic data from which to read the RTV data. <P1> is the Mx310 to Mx363, to be assigned with the data, <P2> is the slot offset from where it reads the low part 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: M310 to M373, or MF310 to MF373, or ML310 to ML341 or MD310 to MD341		
		2	Defines the slot offset in the TransnET cyclic data for the read RTV register, lower part.		
		3	Defines the slot offset in the TransnET cyclic data for the read RTV register, higher part. Optional, only available for ML or MD		
*ASW[.<AXIS>] = <P1>, <P2>[, <P3>]	232		Sets the slot (offset in the TransnET cyclic data) where the controller writes cyclic data RTV		
		1	Defines the register to be written 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 cyclic data where the controller writes the register lower part		
		3	Defines the slot in the TransnET cyclic data where the controller writes the register, MSL part. Only available for 64bits registers		
*CH_BIT_REG32[.<AXIS>] = <P1>, <P2>, <P3>	90		Allows writing any bits (set bits and/or clear bits) given by mask <P2> with defined value <P3> of any 32 bit integer register <P1> without modifying the other bits in the register		
		1	Destination register		
		2	Mask of the bits that can be modified		
		3	Mask value of the bits		
*CHKDISTGRT = <P1>, <P2>, <P3>	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.<AXIS >= <P1>	180		Clear busy state due to a wait command. Currently – pending the resolution of a known issue - <P1> is ignored and the commands clears the wait command sent by the same application interface that sends this CLRWAIT command.		
		1	mask of the entry where the wait could be cleared (Required but currently ignored)	Bit 1	CSM thread 1
				Bit 2	CSM thread 2
				Bit 3	CSM thread 3
				Bit 16	Application bus PCI
				Bit 17	Application bus TCP/IP Port 1

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
				Bit 18	Application bus TCP/IP Port 2
				Bit 19	Application bus TCP/IP Port 3
*CLX[.<AXIS>] = <P1>	17		Clears all user variables (X, XF, XL, XD, Y register depending <P1>)		
		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
*END[.<AXIS>] = <P1>	0		Stops the CSM thread defined by <P1> or stops all threads if command without <P1>		
		1	Thread to stop		
*ERR[.<AXIS>][= <P1>]	80		Puts ULTIMET ADVANCED in error (error number (M64) = <P1>)		
		1	Error number, which will appear in *M64		
*ESMRTVRSVD = <P1>, <P2>, <P3>	14340		Indicates to the ESM core RTV manager that register <p1> on axis <p2> is not to be used in the automatic RTV monitoring assignment, typically because it is required to be reserved by the application.		
		1	Register Specification		
		2	Axis Number		
		3	Reserved/Unreserved	0	Unreserve
				1	Reserve
*ESMSUBACT = <P1>, <P2>	14338		Switches submodule indexed <p1> into full operation, typically starting high priority activities, and required power on… This must be the last step of ESM enabling.		
		1	Submodule index		
		2	On/Off	0	Disable
				1	Enable
*ESMSUBEN = <P1>, <P2>	14337		Performs the “dynamic” initialisations like RTV setup of submodule indexed <p1>. This must be the first step of ESM enabling.		
		1	Submodule index		
		2	On/Off	0	Disable
				1	Enable
*ESUBNAME = <P1>	14336		Get the name of submodule indexed <p1> in *M442.		
		1	Submodule index		
*GETDINS = <P1>, <P2>	738		Get the actual value of a specified digital local input		

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
		1	Index of the digital local input to get		
		2	Value of the input		
*GETDOUT = <P1>, <P2>	735		Get the value of a specified digital local output		
		1	Index of the digital local output to get		
		2	Value of the output		
*GETMDINS = <P1>, <P2>, <P3>	734		Get the state of a set (mask) of local digital inputs		
		1	First input bit of the mask		
		2	Number of input bits of the mask		
		3	Value of the masked input		
*GETMDOUT = <P1>, <P2>, <P3>	732		Get the state of a set (mask) of local digital outputs		
		1	First output bit of the mask		
		2	Number of bits of the mask		
		3	Value of the masked output		
*HLB[.<AXIS>]	121		Stops the CSM threads, stops the interpolated movements with a defined deceleration and disables the interpolation mode at the end of the movement brake		
*HLO[.<AXIS>]	119		Stops the CSM threads, stops the interpolated movements with an infinite deceleration and disables the interpolation mode		
*HLT[.<AXIS>]	120		Stops the CSM threads, stops the interpolated movements with an infinite deceleration and disables the interpolation mode		
*IABSCOORDS = <P1>, ..., <P5>	556		Defines the coordinates of the current position of the axes of the selected interpolation group as the new origin of the coordinates frame		
		1	Interpolation group		
		2	X axis abs coordinate		
		3	Y axis abs coordinate		
		4	Z axis abs coordinate		
		5	Theta axis abs coordinate		
*IABSMODE = <P1>, <P2>	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 absolute reference mode		
*IBEGIN = <P1>	553		Switches the selected interpolation group into interpolated mode		
		1	Interpolation group		
*IBRK = <P1>	653		Stops the selected interpolation group, with the current deceleration value		
		1	Interpolation group		

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
*ICCW = <P1>, ..., <P5>	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>	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>	657		Clears the interpolation command buffer of the selected interpolation group		
		1	Interpolation group		
*ICONC = <P1>	1030		Starts the concatenated mode in the selected interpolation group		
		1	Interpolation group		
*ICONT = <P1>	654		Restarts the previously stopped interpolation group, with the current acceleration value		
		1	Interpolation group		
*ICW = <P1>, ..., <P5>	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>	1026		Adds a 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		
*IEND = <P1>	554		Switches the selected interpolation group out of interpolated mode		
		1	Interpolation group		
*ILINE = <P1>, ..., <P5>	1025		Adds a linear move to the trajectory for the selected interpolation group		

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
		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>	1044		Locks the selected interpolation group for the selected interpolation group		
		1	Interpolation group		
*IMARK = <P1>, ..., <P5>	1039		Adds a mark command 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		
		5	Second parameter associated to the action		
*IMRES = <P1>	1063		Restore the default matrix of the selected interpolation group		
		1	Interpolation group		
*IMROT = <P1>, <P2>, <P3>	1056		Rotates the axes plane of the selected interpolation group		
		1	Interpolation group		
		2	Rotation plane	0	X-Y rotation plane

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
				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>	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>	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>	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		
*INCONC = <P1>	1031		Quits the concatenated mode in the selected interpolation group		
		1	Interpolation group		
*IND = <P1>, [<P2>]	45		Starts a homing sequence on a group of axes		
		1	Axis mask		
		2	Disable adjustment movement (disable = 255)		
*IOCFGRST	595		Resets IO pins configuration to "none used"		

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
*IPT = <P1>, …, <P6>	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>	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>	662		Updates the Y registers for PVT trajectory modifications		
		1	Depth to copy to depth 0		
		2	Mask of the Y registers to copy		
*ISET = <P1>, ..., <P5>	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>	656		Stops the movement of the specified interpolation group with an infinite deceleration		
		1	Interpolation group		
*ITACC = <P1>, <P2>	1036		Sets tangential acceleration of the selected interpolation group		
		1	Interpolation group		

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
		2	Tangential acceleration value		
*ITANSPDMASK = <P1>, <P2>	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>	1037		Sets tangential deceleration of the selected interpolation group		
		1	Interpolation group		
		2	Tangential deceleration value		
*ITJRT = <P1>, <P2>	1038		Sets tangential jerk time of the selected interpolation group		
		1	Interpolation group		
		2	Tangential jerk time		
*ITSPD = <P1>, <P2>	1035		Sets tangential speed of the selected interpolation group		
		1	Interpolation group		
		2	Tangential speed value		
*IUNLOCK = <P1>	655		Starts the selected interpolation group by unlocking the 'ILOCK' function		
		1	Interpolation group		
*IWTT = <P1>, <P2>	1029		Waits for a defined time in interpolated mode (selected interpolation group)		
		1	Interpolation group		
		2	Waiting time value		
*JMP[.<AXIS>] = <P1>, <P2>	26		Start CSM high level function <P1> in thread number <P2>		
		1	CSM high level function number		
		2	Thread number		
*NEW[.<AXIS>] = <P1>	78		Restores the default registers values (K and W), clears registers (X, Y and L) in ram memory according to <P1>		
		1	Register selection	0	Clears X user variables, Y registers, sets default values for K and W parameters in ram memory
				2	Sets default values for K and W 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
				10	Clears ESM W registers in ram memory

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
*RES[.<AXIS>] = <P1>	49		Restores the persistent storage saved registers (K, W, X, Y and L) in ram memory according to <P1>		
		1	Register selection	0	Restores L look-up tables, K and W parameters and X user variables and Y registers from persistent storage to ram memory
				1	Restores L look-up tables from persistent storage to ram memory
				2	Restores K parameters, W parameters, X user variables and Y registers from persistent storage to ram memory
				3	Restores K parameters from persistent storage to ram memory
				5	Restores X user variables and Y registers from persistent storage to ram memory
				6	Restores L look-up tables from persistent storage to ram memory
				10	Restores ESM W parameter from persistent storage to RAM
*RST[.<AXIS>]	79		Resets the error flag of ULTIMET ADVANCED		
*RSTDOUT = <P1>	737		Reset a specified digital local output		
		1	Index of the digital local output to reset		
*RSU = <P1>	601		Perform a reset hardware on the UltimET		
		1	Security password		
*SAV[.<AXIS>] = <P1>	48		Saves the current registers (K, W, X, Y and L) in persistent storage in function of <P1>		
		1	Register selection	0	Saves L look-up tables, K and W parameters, X user variables and Y registers in persistence storage
				1	Saves L look-up tables in persistent storage
				2	Saves K parameters, W parameters, X user variables and Y registers in persistent storage
				3	Saves K parameters in persistent storage
				5	Saves X user variables and Y registers in persistent storage
				6	Saves L look-up tables in persistent storage
				10	Saves ESM W parameter in persistent storage

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
*SETRANGE = <P1>, …., <P128>	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>	736		Set a specified digital local output		
		1	Index of the digital local output to set		
*SETMDOUT = <P1>, <P2>, <P3>	733		Apply a mask on local digital outputs		
		1	First output bit of the mask		
		2	Number of output bits of the mask		
		3	Value to put on masked digital output		
*SETRTV = <P1>, <P2>	532		Sets an RTV slot P1 with a P2 value		
		1	Slot number		
		2	Value		
*SLT = <P1>	27		Set Local Time		
		1	Epoch time (seconds from 1 jan 1970)		
*SPIABORT	579		Aborts spi transfer if any		
*SPIBURSTCFG = <P1>, <P2>	586		Configure SPI burst transfer, defining RW mode <P1> and the amount of transfered data in bytes <P2>		
		1	RW mode		
		2	Number of bytes		
*SPICON = <P1>, ..., <P5>	580		Connnect and configure spi interace		
		1	Spi Mode (Master=1, Slave=0)		
		2	Frequency rate	2	SPI clock frequency: 25MHz
				3	SPI clock frequency: 12.5MHz
				4	SPI clock frequency: 6.25MHz
				5	SPI clock frequency: 3.125MHz
				6	SPI clock frequency: 1.5625MHz
				7	SPI clock frequency: 781.25kHz
		3	Clock mode	0	SPI mode: CPOL=0, CPHA=0
				1	SPI mode: CPOL=0, CPHA=1
				2	SPI mode: CPOL=1, CPHA=0

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
				3	SPI mode: CPOL=1, CPHA=1
		4	CRC mode	0	SPI CRC mode: no CRC
				1	SPI CRC mode: CRC-8 CCITT
		5	Bit order (MSB=0 or LSB=1)		
*SPIDIS	581		Disconnect spi interface and free associated Inputs/Outputs		
*SPIREAD = <P1>, <P2>, [<P3>[, <P4>[, <P5>]]]	582		Reads <P1> bytes of data on SPI interface		
		1	Number of bytes		
		2	Read data 1		
		3	Read data 2		
		4	Read data 3		
		5	Read data 4		
*SPIRST	589		Reset SPI configuration		
*SPIRW = <P1>, <P2>, <P3>, <P4>, <P5>, <P6>, [<P7>[, <P8>[, <P9>]]]	584		Write and read <P1> bytes of data on SPI interface		
		1	Number of bytes		
		2	Data to write 1		
		3	Data to write 2		
		4	Data to write 3		
		5	Data to write 4		
		6	Read data 1		
		7	Read data 2		
		8	Read data 3		
		9	Read data 4		
*SPIWRITE = <P1>, <P2>, [<P3>[, <P4>[, <P5>]]]	583		Write <P1> bytes of data on SPI interface		
		1	Number of bytes		
		2	Data to write 1		
		3	Data to write 2		
		4	Data to write 3		
		5	Data to write 4		
*SSR[.<AXIS>] = <P1>, <P2>, [<P3>, <P4>[, <P5>, <P6>[, <P7>, <P8>[, <P9>, <P10>[, <P11>, <P12>]]]]]	127		Set Several Registers 1 up to 6		

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
		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)		
		11	6th register and depth definition (optional parameter)		
		12	6th register (optional parameter)		
*STARTCSM = <P1>	224		Starts a customer software module		
		1	csm number		
*STOPCSM	225		Stops a customer software module		
*WAITCSM = <P1>	238		Wait CSM started (wait for: default condition: CSM running in REMOTE_CONTROLLED mode, <PAR1> is optional: 0x1: CSM running in REMOTE_CONTROLLED mode, 0x2: CSM connection established, 0x4: CSM Core running)		
		1	Wait Condition (0x1: CSM running in REMOTE_CONTROLLED mode, 0x2: CSM connection established, 0x4: CSM Core running)	1	CSM running in REMOTE_CONTROLLED mode (default value)
				2	CSM Connection established
				4	CSM Core running
*WBC[.<AXIS>] = <P1>, <P2>	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>	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>	53		Waits until <P1> becomes greater than <P2>		
		1	Value or register to test		
		2	Reference value or register		
*WPL[.<AXIS>] = <P1>, <P2>	52		Waits until <P1> becomes lower than <P2>		

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
		1	Value or register to test		
		2	Reference value or register		
*WRST	81		Reset remanent Warnings		
*WSBC = <P1>, <P2>	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>	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>	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>	513		Waits for the specified mark <P2> of the defined interpolation group <P1>		
		1	Interpolation group		
		2	Mark number		
*WTM[.<AXIS>] = <P1>	8		Waits for the end of interpolated movement of the group defined by <P1>		
		1	Interpolation group		
*WTT[.<AXIS>] = <P1>	10		Waits for the specified amount of time		
		1	Time		
*<REG>[.<AXIS>] = <P2>	123		Assigns the value of <P2> to <REG>. This is available with all kinds of registers excepted monitoring register (M)		
		1	Destination register		
		2	Source value or register		
*<REG>[.<AXIS>] += <P2>	91		Adds <P2> to <REG> and puts the result in <REG>		
		1	Destination register		
		2	Value or register to add		
*<REG>[.<AXIS>] &= <P2>	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>	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		

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
*<REG>[.<AXIS>] = <P2>	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>	94		Divides <REG> by <P2> and puts the result in <REG>		
		1	Destination register		
		2	Value or register to divide by		
*<REG>[.<AXIS>] %= <P2>	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>	93		Multiplies <REG> by <P2> and puts the result in <REG>		
		1	Destination register		
		2	Value or register to multiply with		
*<REG>[.<AXIS>] ~= <P2>	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>	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>	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		
*<REG>[.<AXIS>] <<= <P2>	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>	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		

Syntax	Cmd nb	Param. nb	Command or parameter description	Values or bit#	Value description
*<REG>[.<AXIS>] -= <P2>	92		Substracts <P2> from <REG> and puts the result in <REG>		
		1	Destination register		
		2	Value or register to substract		
*<REG>[.<AXIS>] ^= <P2>	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>	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>	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		

30. K parameters list

K	Alias	Values or bit#	Description	Default value	Minimum value	Maximum value
*K15			ESM submodules mask to start	0	0	31
*K16			CSM and JMP number to start automatically	0	0	INT32_MAX
*K33			Enables input mode	0	0	255
		Bit 4	Enables external error if the DIN match with the mask K305			
*K120			Enables trace synchronization mode	0	0	1
*K121			Trace register selection type (depth 0 for trace 0; depth 1 for trace 1,...)	0	0	255
		1	User variables register (X)			
		2	Parameter register (K)			
		3	Monitoring register (M)			
		8	Look-up tables register (L)			
		10	ESM monitoring register (R)			
		11	Advanced user register (Y)			
		12	ESM parameter register (W)			
		33	User variables register (XF)			
		34	Parameter register (KF)			
		35	Monitoring register (MF)			
		42	ESM monitoring register (RF)			
		44	ESM parameter register (WF)			
		65	User variables register (XL)			
		66	Parameter register (KL)			
		67	Monitoring register (ML)			
		74	ESM monitoring register (RL)			
		76	ESM parameter register (WL)			
		97	User variables register (XD)			
		98	Parameter register (KD)			
		99	Monitoring register (MD)			
		106	ESM monitoring register (RD)			
		108	ESM parameter register (WD)			
*K122			Trace register number and depth selection (bit#0-15 for register number, bit#16-23 for depth)	0	0	0
*K123			Trace sampling time selection (period of 25 us)	4	4	INT32_MAX
*K124			Trace trigger edge selection	0	-1	1

K	Alias	Values or bit#	Description	Default value	Minimum value	Maximum value
*K125			Trace: trigger mode selection	0	0	8
		0	Immediate			
		1	Start of movement			
		2	End of movement			
		3	Trigger at a given position			
		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			
*K126			Definition of the trigger parameter according to the trigger mode	0	0	INT32_MAX
*K127			Definition of the trigger level according to the trigger mode	0	0	INT32_MAX
*KD127			Definition of the trigger level according to the trigger mode	0	FLOAT_MIN	FLOAT_MAX
*KF127			Definition of the trigger level according to the trigger mode	0	FLOAT_MIN	FLOAT_MAX
*KL127			Definition of the trigger level according to the trigger mode	0	INT64_MIN	INT64_MAX
*K128			Number of points to acquire for the traces	16384	0	16384
*K129			Pre or post trigger value	0	INT32_MIN	INT32_MAX
*K140			Error propagation checked group mask	0	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			Error propagation published group mask	0	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			
*K144			Thread mask not stopped by HLT HLB HLO. Bit#1 for thread 1, bit#2 for thread 2, bit#3 for thread 3	0	0	255
*K166	WARNMSK		Warning mask	0	0	UINT32_MAX
		Bit 20	Interpolation buffer nearly full.			
*K171	DOUT		Digital outputs	0	0	255
*K177			User status	0	0	0

K	Alias	Values or bit#	Description	Default value	Minimum value	Maximum value
*KL257			Selected licensed features mask (enabled: bit = 1)	0	INT64_MIN	INT64_MAX
*K297			Mask for active thread information on M63 status (bit#1 thread1, bit#2 thread2, bit#3 thread3)	0	0	255
*K305			Requested DIN mask for external error (bit 0-15: state 1, bit 16-31: state 0)	0	0	255
*K353			Requested DOUT mask for TransnET cycle	0	0	255
*K357			Requested DOUT mask for security management (bit 0-15: state 1, bit 16-31: state 0)	0	0	255
*K360			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
*K513	IBUFWNG		Interpolation: threshold of the percent of command buffer size of remaining free slots that generates a warning	30	0	100
*K522			Interpolation, unit position factor. The depths correspond to the ipol group numbers	1	1	INT32_MAX
*K523			Interpolation, unit position exponent. The depths correspond to the ipol group numbers	-8	-16	0
*K524			Interpolation, unit speed exponent. The depths correspond to the ipol group numbers	-6	-10	0
*K525			Interpolation, unit acceleration exponent. The depths correspond to the ipol group numbers	-6	-10	0
*K526			Interpolation, unit time exponent. The depths correspond to the ipol group numbers	-6	-10	0
*K530	ISPDRATE		Interpolation, speed rate (unit: %). The depths correspond to the ipol group numbers	100	0	100
*K540			Interpolation, mark number. The depths correspond to the ipol group numbers	0	0	INT32_MAX
*K541			Interpolation, DOUT reservation for the marks	0	0	15
*K546			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			Linked to IPVTUPDATE command: stores the first number of a block of 32 Y registers that can be updated	0	0	224
*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	1	1	INT32_MAX
*K580			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			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			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		Control of user's LED (1=on, 0=off. Depth 0 = green led, depth 1 = red led)	0	0	1
*K600			Sets the interrupt refresh frequency for interpolation	0	0	16
*K631			Action to take in case of error	3	0	3
*K707			HLX CSM abort mode	0	0	1
*K708			ULTIMET ADVANCED, Aux com X4 IP address configuration	-1062731262	0	0
*K717	ICAM		Interpolation, came value (unit: %). The depths correspond to the ipol group numbers	100	1	100
*K800			SPI communication, clock digital IO number mask	0	0	128
*K801			SPI communication, MISO digital IO number mask	0	0	128
*K802			SPI communication, MOSI digital IO number mask	0	0	128

K	Alias	Values or bit#	Description	Default value	Minimum value	Maximum value
*K803			SPI communication, Slave Select (SS) digital IO number mask	0	0	128
*K804			Local IO management, mask of digital IO used as digital inputs (DIN)	0	0	255
*K805			Local IO management, requested mask of digital IO used as digital outputs (DOUT)	0	0	255
*K811			SPI CRC error mode	0	0	2

31. M monitoring registers list

M	Alias	Values or bit#	Description
*M50	DIN		Digital inputs
*M63	SDC		ULTIMET ADVANCED 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 ADVANCED is executing an interpolated trajectory
		Bit 23	ULTIMET ADVANCED is in warning mode
		Bit 24	ULTIMET ADVANCED is executing a CSM
		Bit 25	ULTIMET ADVANCED is executing an interpolated trajectory, group 0
		Bit 26	ULTIMET ADVANCED is in error mode
		Bit 27	Trace busy flag is set during a register trace acquisition
		Bit 29	ULTIMET ADVANCED is executing an interpolated trajectory, group 1
		Bit 30	TransnET connection has been interrupted
		Bit 31	CSM Thread (*K297) is running
*M64	ERRCD		Gives the error code
*M66	WARCD		Warning code
*M70	CTYP		Indicates the type of ULTIMET ADVANCED
*M71			Gives the software boot version of ULTIMET ADVANCED. A special ETEL procedure allows the conversion of this value in the software boot version (format is similar to M72)
*M72	VER		Gives the firmware version of the ULTIMET ADVANCED. A special ETEL procedure allows the conversion of this value in the firmware version
*M73	SER		Gives the serial number of the ULTIMET ADVANCED
*M84	ESMNAME		Gives the name of the ESM currently loaded.
*M85	ARTICLE		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			Gives the ULTIMET ADVANCED axis number
*M90	CTEMP		Gives the temperature of the device measured by a thermostat on the heat sink
*M92			FTP feature state
*M95			User function Index running in the thread
*M97			Mask of the sequence threads currently active
*M100			CSM activity status: bit#0: CSM activity in REMOTE_CONTROLLED mode: 1 running, 0 not running; bit#1: CSM-UA connection state: 1: established, 0: none; bit#2: CSM core#3 status: 1: running, 0: idle state

M	Alias	Values or bit#	Description
*M120			Traces synchronization mode
*M121			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)
		10	ESM monitoring register (R)
		11	Advanced user register (Y)
		12	ESM parameter register (W)
		33	User variables register (XF)
		34	Parameter register (KF)
		35	Monitoring register (MF)
		42	ESM monitoring register (RF)
		44	ESM parameter register (WF)
		65	User variables register (XL)
		66	Parameter register (KL)
		67	Monitoring register (ML)
		74	ESM monitoring register (RL)
		76	ESM parameter register (WL)
		97	User variables register (XD)
		98	Parameter register (KD)
		99	Monitoring register (MD)
		106	ESM monitoring register (RD)
		108	ESM parameter register (WD)
*M122			Trace register number and depth selection (bit#0-15 for register number, bit#16-23 for depth)
*M123			Trace sampling time (period of 25us)
*M124			Trace trigger edge
*M125			Trace trigger mode
		0	Immediate
		1	Start of movement
		2	End of movement
		3	Trigger at a given position
		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
*M126			Definition of the trigger parameter according to the trigger mode
*M127			The value gives the trigger level according to the trigger mode
*MD127			The value gives the trigger level according to the trigger mode
*MF127			The value gives the trigger level according to the trigger mode
*ML127			The value gives the trigger level according to the trigger mode
*M128			Number of acquired points for the trace
*M129			Pre or post trigger value
*M131			Traces acquisition state

M	Alias	Values or bit#	Description
		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
		5	Acquisition state: End of acquisition
*M134			Freezes time of the measure
*ML134			Freezes time of the measure
*M135			Number of the T register with the trigger condition
*M136			Number of available points
*M141			TransnET error groups status
*M166	WARNFLG		Warning flags
		Bit 20	Interpolation buffer nearly full.
*M171			Gives the state of the digital outputs
*M203			ULTIMET ADVANCED, Ethernet X1 MAC address
*M204			ULTIMET ADVANCED, Ethernet X2 MAC address
*M205			ULTIMET ADVANCED, Ethernet X1 IP config
*M206			ULTIMET ADVANCED, Ethernet X2 IP config
*M228			Reference time counter in unit of 25 us. This counter runs freely and is synchronized with UltimET when TransnET is connected
*ML228			Reference time counter in unit of 25 us. This counter runs freely and is synchronized with UltimET when TransnET is connected
*M255			Is the licensable features grace period active?
*ML255			Licensable features mask (existing in Firmware: bit = 1)
*M256			Licensable features grace period down-counter [min]
*ML256			Licensed features mask (enabled: bit = 1)
*ML257			Active licensed features mask (enabled: bit = 1)
*M260			Status of SPI communication
*M261			Data received from SPI communication
*M262			SPI CRC error counter
*M263			SPI configuration
		Bit 0	Master or Slave configuration for SPI. Bit = 0 for Slave, Bit = 1 for Mater
		Bit 1..2	Clock mode configuration (CPOL, CPHA): 0 (0,0), 1 (0, 1), 2 (1,0), 3(1,1)
		Bit 3	WORD configuration. Bit = 0 for MSB first, Bit = 1 for LSB first.
		Bit 4..6	Clock frequency. 010: 25MHz, 011:12.5MHz ... lowest is 111: 0.78125MHz
		Bit 16	Enable CRC. Bit = 0 disable, Bit = 1 enable
		Bit 17..19	CRC size. 001: CRC-8, 010: CRC-16, 011: CRC-32
		Bit 22..23	CRC error management. 00: increase CRC error counter M262, 01: re-asks the SPI data and in case of a 2nd error increase CRC error counter M262
*M282			Registers or sequence version (depth 0 to 2: not used, depth 3 to 7 and 10 as for SAV command)
*M284			CSM Name
*M285			CSM Description
*M286			CSM compilation platform
*M287			CSM compilation mode
*M305			Actual DIN mask for external error (bit 0-15: state 1, bit 16-31: state 0)
*M310			RTV monitoring
*MD310			RTV monitoring
*MF310			RTV monitoring

M	Alias	Values or bit#	Description
*ML310			RTV monitoring
*M311			RTV monitoring
*MD311			RTV monitoring
*MF311			RTV monitoring
*ML311			RTV monitoring
*M312			RTV monitoring
*MD312			RTV monitoring
*MF312			RTV monitoring
*ML312			RTV monitoring
*M313			RTV monitoring
*MD313			RTV monitoring
*MF313			RTV monitoring
*ML313			RTV monitoring
*M314			RTV monitoring
*MD314			RTV monitoring
*MF314			RTV monitoring
*ML314			RTV monitoring
*M315			RTV monitoring
*MD315			RTV monitoring
*MF315			RTV monitoring
*ML315			RTV monitoring
*M316			RTV monitoring
*MD316			RTV monitoring
*MF316			RTV monitoring
*ML316			RTV monitoring
*M317			RTV monitoring
*MD317			RTV monitoring
*MF317			RTV monitoring
*ML317			RTV monitoring
*M318			RTV monitoring
*MD318			RTV monitoring
*MF318			RTV monitoring
*ML318			RTV monitoring
*M319			RTV monitoring
*MD319			RTV monitoring
*MF319			RTV monitoring
*ML319			RTV monitoring
*M320			RTV monitoring
*MD320			RTV monitoring
*MF320			RTV monitoring
*ML320			RTV monitoring
*M321			RTV monitoring
*MD321			RTV monitoring
*MF321			RTV monitoring
*ML321			RTV monitoring
*M322			RTV monitoring

M	Alias	Values or bit#	Description
*MD322			RTV monitoring
*MF322			RTV monitoring
*ML322			RTV monitoring
*M323			RTV monitoring
*MD323			RTV monitoring
*MF323			RTV monitoring
*ML323			RTV monitoring
*M324			RTV monitoring
*MD324			RTV monitoring
*MF324			RTV monitoring
*ML324			RTV monitoring
*M325			RTV monitoring
*MD325			RTV monitoring
*MF325			RTV monitoring
*ML325			RTV monitoring
*M326			RTV monitoring
*MD326			RTV monitoring
*MF326			RTV monitoring
*ML326			RTV monitoring
*M327			RTV monitoring
*MD327			RTV monitoring
*MF327			RTV monitoring
*ML327			RTV monitoring
*M328			RTV monitoring
*MD328			RTV monitoring
*MF328			RTV monitoring
*ML328			RTV monitoring
*M329			RTV monitoring
*MD329			RTV monitoring
*MF329			RTV monitoring
*ML329			RTV monitoring
*M330			RTV monitoring
*MD330			RTV monitoring
*MF330			RTV monitoring
*ML330			RTV monitoring
*M331			RTV monitoring
*MD331			RTV monitoring
*MF331			RTV monitoring
*ML331			RTV monitoring
*M332			RTV monitoring
*MD332			RTV monitoring
*MF332			RTV monitoring
*ML332			RTV monitoring
*M333			RTV monitoring
*MD333			RTV monitoring
*MF333			RTV monitoring

M	Alias	Values or bit#	Description
*ML333			RTV monitoring
*M334			RTV monitoring
*MD334			RTV monitoring
*MF334			RTV monitoring
*ML334			RTV monitoring
*M335			RTV monitoring
*MD335			RTV monitoring
*MF335			RTV monitoring
*ML335			RTV monitoring
*M336			RTV monitoring
*MD336			RTV monitoring
*MF336			RTV monitoring
*ML336			RTV monitoring
*M337			RTV monitoring
*MD337			RTV monitoring
*MF337			RTV monitoring
*ML337			RTV monitoring
*M338			RTV monitoring
*MD338			RTV monitoring
*MF338			RTV monitoring
*ML338			RTV monitoring
*M339			RTV monitoring
*MD339			RTV monitoring
*MF339			RTV monitoring
*ML339			RTV monitoring
*M340			RTV monitoring
*MD340			RTV monitoring
*MF340			RTV monitoring
*ML340			RTV monitoring
*M341			RTV monitoring
*MD341			RTV monitoring
*MF341			RTV monitoring
*ML341			RTV monitoring
*M342			RTV monitoring
*MF342			RTV monitoring
*M343			RTV monitoring
*MF343			RTV monitoring
*M344			RTV monitoring
*MF344			RTV monitoring
*M345			RTV monitoring
*MF345			RTV monitoring
*M346			RTV monitoring
*MF346			RTV monitoring
*M347			RTV monitoring
*MF347			RTV monitoring
*M348			RTV monitoring

M	Alias	Values or bit#	Description
*MF348			RTV monitoring
*M349			RTV monitoring
*MF349			RTV monitoring
*M350			RTV monitoring
*MF350			RTV monitoring
*M351			RTV monitoring
*MF351			RTV monitoring
*M352			RTV monitoring
*MF352			RTV monitoring
*M353			RTV monitoring
*MF353			RTV monitoring
*M354			RTV monitoring
*MF354			RTV monitoring
*M355			RTV monitoring
*MF355			RTV monitoring
*M356			RTV monitoring
*MF356			RTV monitoring
*M357			RTV monitoring
*MF357			RTV monitoring
*M358			RTV monitoring
*MF358			RTV monitoring
*M359			RTV monitoring
*MF359			RTV monitoring
*M360			RTV monitoring
*MF360			RTV monitoring
*M361			RTV monitoring
*MF361			RTV monitoring
*M362			RTV monitoring
*MF362			RTV monitoring
*M363			RTV monitoring
*MF363			RTV monitoring
*M364			RTV monitoring
*MF364			RTV monitoring
*M365			RTV monitoring
*MF365			RTV monitoring
*M366			RTV monitoring
*MF366			RTV monitoring
*M367			RTV monitoring
*MF367			RTV monitoring
*M368			RTV monitoring
*MF368			RTV monitoring
*M369			RTV monitoring
*MF369			RTV monitoring
*M370			RTV monitoring
*MF370			RTV monitoring
*M371			RTV monitoring

M	Alias	Values or bit#	Description
*MF371			RTV monitoring
*M372			RTV monitoring
*MF372			RTV monitoring
*M373			RTV monitoring
*MF373			RTV monitoring
*M439	ESMSUBSTS		ESM Sub module status
*M440	ESMSUB-COUNT		Number of ESM Submodules
*M441			ESM Submodules signatures (id,version)
*M442			ESM Submodules name
*M443			Number of possible RTV write slots (ASW)
*M444			Count of used monitorings for ASR by ESM
*M445			List of used monitorings for ASR by ESM
*M447			Number of possible RTV read slots (ASR)
*M448			Used RTV slots (by ASW)
*M449			Number of written RTV (by ASW command)
*M511	DWNSTS		Download Error Code
*M512	INBUFFRE		Interpolation: remaining free slots in interpolation commands buffer
*ML512			Mask of the devices present on the TransnET
*M513	IBUFNMIN		Interpolation: actual threshold of the number of remaing free slots in command buffer that generates a warning
*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 status (depth 0=group 0…).
		Bit 0	Active state.
		Bit 1	Moving state.
		Bit 2	Emergency brake state.
*M524			Number of free RTV slots. These RTV slots can be managed using, get or free slot commands
*M525			First RTV slot number
*ML525			Interpolation, interpolated axis mask (depths 0 = axis X group 0, 1=Y group 0…, depth 4 = axis X group 1...)
*M526			Gives the number of RTV slots
*M530			Interpolation group is active (1) or not (0). The depths correspond to the ipol group numbers
*M531			Interpolation buffer locked (1) or not (0). The depths correspond to the ipol group numbers
*M535			Number of commands inside interpolation buffer. The depths correspond to the ipol group numbers
*M541			Local IO, actual mask of IO used for interpolation
*M590			Interpolation, absolute coordinates mode: absolute (1) or relative (0). The depths correspond to the ipol group numbers
*M629			TransnET frame error counter
*M630			TransnET operational state
*ML630			Mask of all the error bits of the nodes present on the TransnET
*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)

M	Alias	Values or bit#	Description
*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)
*M649			Tangential velocity, ipol group 0 (unit: ufsi)
*M650			Tangential acceleration, ipol group 0 (unit: ufai)
*M651			Tangential deceleration, ipol group 0 (unit: ufai)
*M652			Tangential jerk time, ipol group 0 (unit: ufti)
*M658			Maximal time value for interpolation commands (unit: ufti)
*M659			Maximal distance for interpolation commands (unit: ufpi)
*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)
*M669			Tangential velocity, ipol group 1 (unit: ufsi)
*M670			Tangential acceleration, ipol group 1 (unit: ufai)
*M671			Tangential deceleration, ipol group 1 (unit: ufai)
*M672			Tangential jerk time, ipol group 1 (unit: ufti)
*M680	TANSPD		Current interpolation tangential speed. The depth (0 and 1) correspond to the interpolation group number
*MF680	TANSPEED		Current interpolation tangential speed. The depth (0 and 1) correspond to the interpolation group number
*M681	TANSPDMASK		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
*M707			ULTIMET ADVANCED, Aux com X4 MAC address
*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...)
*M804			Local IO management, mask of digital IO used as digital inputs (DIN)
*M805			Local IO management, actual mask of digital IO used as digital outputs (DOUT)
*M806			Local IO, mask of all input IO pins
*M807			Local IO, mask of all output IO pins
*M808			Actual DOUT mask for TransnET cycle
*M809			Actual DOUT mask for security management (bit 0-15: state 1, bit 16-31: state 0)

32. Warnings reference list

This error code is given by the monitoring *M66.

*M66	Description	Solutions
5	Stream: Unexpected end of stream received.	General case: - Stream usage is not correct. Data volume incorrectly calculated. Please contact your provider.
8	No license file. Grace period activated due to active licensable features	General case: - Wrong value coming from the digital Hall sensor
9	New License file is not valid. Previous one is kept.	General case: - Lost synchronization between TransnET and controller interrupts.
11	Command Flow Manager list is full.	General case: - Reduce the command flow sent to the UltimET Advanced.
15	Error using flash IP config - using defaults.	General case: - Error in the UltimET Advanced production configuration. Contact your provider.
18	Interpolation: emergency brake has occurred.	General case: - An emergency brake has occurred, because the velocity of the last PVT segment was not zero.
20	Interpolation buffer nearly full.	General case: - Use iLock/iUnlock method to regulate the buffer of the interpolation.
26	FW update already running.	General case: - Ensure no multiple firmware downloads are ongoing over different interfaces.
27	FW update not started.	General case: - Please contact your provider.
28	FW update: nothing downloaded.	General case: - Please contact your provider.
29	FW update: End started with an error notified.	General case: - Check the speed of your communication interface. Too slow communication between PC and controller may raise this warning.
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.
31	Unable to read flash.	General case: - Unable to read the flash. Contact your provider.

33. Errors reference list

This error code is given by the monitoring *M64.

33.1 Firmware error codes

*M64	Description	Solutions
1000	Communication error. TransnET down	General case: - Communication error: The bus has not been initialized properly.
1040	Command requires synchronized Tx but buses are not synchronized or not broadcastable	General case: - Command management error: Please contact your provider.
1041	Received command is not implemented in system	General case: - Command management error: Review the command ID (command number and recored number) sent to the controller. The current command ID is not valid.
1042	Received command cannot take remote indirect parameters in this category	General case: - Command management error: The command sent to the controller does allow a parameter for another controller.
1043	Timeout waiting for indirect remote parameter value	General case: - Command management error: A command parameter cannot be resolved. Check that this parameter is addressed to an existing controller.
1044	Timeout dispatching command	General case: - Command management error: a controller is not accessible for this command. Please contact your provider.
1045	Timeout waiting for command execution acknowledgment	General case: - Command management error: command acknowledge is not done within defined timeout.
1046	The device passed as parameter is not present	General case: - Command management error: The controller addressed in a parameter is not present. Check that this controller is accessible. - Review the command syntax and validate that the parameter addressed controller is correct.
1047	The device is not present	General case: - Command management error: The addressed controller is not present. Check that this controller is accessible. - Review the command syntax (addressed controller).
1048	An incoming message parameter format is not supported	General case: - Command management error: A command parameter format is not supported. Review the command syntax.
1049	An incoming message parameter type is not supported	General case: - Command management error: A command parameter type is not supported. Review the command syntax.
1052	There are too many parameters in the incoming message	General case: - Command management error: The command sent to the controller has too many parameters. Review command syntax.
1053	Syntax error in received command	General case: - Command management error: The syntax of the command sent to the controller is not correct. Review command syntax.
1054	Error adding command in list	General case: - Command management internal error. Please contact your provider.
1055	Command in illegal step	General case: - Command management internal error. Please contact your provider.
1056	Command Management Thread Exception	General case: - Command management internal error. Please contact your provider.
1057	Register index is invalid, either under 0 or greater than the array size	General case: - Command management error: a command parameter index sent to the controller is out of range. Review command syntax.
1058	Register depth is invalid, either under 0 or greater than the allowed number of depths	General case: - Command management error: a command parameter depth sent to the controller is out of range. Review command syntax.
1059	The command parameter format is incorrect (e.g. received Int64 and expected Int32)	General case: - Command management error: A command parameter format is not supported. Review the command syntax.

*M64	Description	Solutions
1060	Invalid filename specified in file stream command	General case: - Stream error: Invalid file name. Please contact your provider.
1062	Bad command record	General case: - Command management internal error. Please contact your provider.
1063	Command not posted	General case: - Command management internal error. Please contact your provider.
1064	Cannot download file type outside firmware download	General case: - Firmware management error: the downloaded file type can be downloaded only a FW download context.
1065	Error getting zipped logfiles size	General case: - Check requested number of files
1070	The command index is not valid for the command table	General case: - Command management error: the syntax (command number, record number) of the command sent to the controller is invalid. Review command syntax.
1080	Application Thread Exception	General case: - Firmware exception. Please contact your provider.
1081	TCPIP Application Server Thread Exception	General case: - Firmware exception. Please contact your provider.
1082	Background Thread Exception	General case: - Firmware exception. Please contact your provider.
1083	High Priority Thread Exception	General case: - Firmware exception. Please contact your provider.
1084	SHM Receive Thread Exception	General case: - Firmware exception. Please contact your provider.
1085	SHM Transmit Thread Exception	General case: - Firmware exception. Please contact your provider.
1086	TCPIP Receive Thread Exception	General case: - Firmware exception. Please contact your provider.
1087	TCPIP Transmit Thread Exception	General case: - Firmware exception. Please contact your provider.
1088	TCPIP Port Properties Server Thread Exception	General case: - Firmware exception. Please contact your provider.
1089	TCPIP Port Properties not completely sent; closing	General case: - Firmware exception. Please contact your provider.
1090	UDP Server Thread Exception	General case: - Firmware exception. Please contact your provider.
1091	PCIe Transmit Thread Exception	General case: - Firmware exception. Please contact your provider.
1092	PCIe Receive Thread Exception	General case: - Firmware exception. Please contact your provider.
1093	Port properties Thread Exception	General case: - Firmware exception. Please contact your provider.
1110	An error occurred while getting the productInfo inside the XML	General case: - Metadata error: controller metadata information is invalid. Update controller firmware and retry - If not successful, contact your provider.
1111	The information from the Metadata doesn't match the information in the FW	General case: - Metadata error: some information present in controller metadata does not match the one present in controller firmware. Update controller firmware and retry - If not successful, contact your provider.
1112	An error occurred while getting the commands inside the XML	General case: - Metadata error: some command information present in controller metadata is invalid. Update controller firmware and retry. - If not successful, contact your provider.
1113	An error occurred while getting the commandItem inside the XML	General case: - Metadata error: some command information present in controller metadata is invalid. Update controller firmware and retry. - If not successful, contact your provider.
1114	An error occurred while getting the commandParameter inside the XML	General case: - Metadata error: some command information present in controller metadata is invalid. Update controller firmware and retry. - If not successful, contact your provider.

*M64	Description	Solutions
1115	An error occurred while getting the commandRecord inside the XML	General case: - Metadata error: some command information present in controller metadata is invalid. Update controller firmware and retry. - If not successful, contact your provider.
1116	The commandID from the Metadata is NULL	General case: - Metadata error: some command information present in controller metadata is invalid. Update controller firmware and retry. - If not successful, contact your provider.
1117	The parsing of the metadata failed, device cannot be created	General case: - Metadata error: controller metadata information is invalid. Update controller firmware and retry. - If not successful, contact your provider.
1118	Error during the creation of the metadata parser	General case: - Metadata error: internal error. Contact your provider.
1119	Bad metadata header	General case: - Metadata error: controller metadata information is invalid. Update controller firmware and retry. - If not successful, contact your provider.
1120	Unhandled metadata header version	General case: - Metadata error: controller metadata information is invalid. Update controller firmware and retry. - If not successful, contact your provider.
1121	Unable to unzip the metadata buffer into memory	General case: - Metadata error: controller metadata information is invalid. Update controller firmware and retry. - Configuration error: check that the controllers have different addresses (set by DIP switch or software). - If not successful, contact your provider.
1122	Unhandled metadata compressed format	General case: - Metadata error: controller metadata information is invalid. Update controller firmware and retry. - If not successful, contact your provider.
1123	No metadata file found	General case: - Metadata error: controller metadata information is not present. Update controller firmware and retry - If not successful, contact your provider.
1200	RTV management. No Free Slot found	General case: - RTV management: Slot has not been get
1202	RTV management. Slot already in use	General case: - RTV management: Slot already in use
1203	SlotNb to free not found	General case: - RTV error: this slot number is not valid, and thus cannot be set free.
1210	ASW Slot not reserved	General case: - RTV management: Bad parameter in the ASW command.
1211	Not enough ASW slot available	General case: - RTV management: ASW slot already assigned.
1220	Not enough ASR slot available	General case: - RTV management: Bad parameter in the ASR command.
1221	Register Type does not fit with number of specified slot	General case: - RTV error: ASR, ASW commands parameters mismatch: the register type (32-bit or 64-bit) does not match the number of slots (1 or 2).
1222	Bad Slot Number specified	General case: - RTV error: first parameter must either be a register or a 0 value to unassign a slot.
1223	First Parameter should be a register address or 0	General case: - RTV error: first parameter must either be a register or a 0 value to unassign a slot.
1224	First Parameter should be a M register address	General case: - RTV error: first parameter must be a monitoring register.
1225	ASR Slot not reserved	General case: - RTV error: one of the slot numbers has not been reserved trough getslot command.
1511	No UA Stream opened for the received chunk	General case: - Stream error: stream upload / download process has not been initialized. Initialization command has not been sent (Start download, start upload…).

*M64	Description	Solutions
1512	Invalid command number for 2E record	General case: - Stream error: invalid stream command number. Review command ID (command number and record number).
1513	Received end of stream chunk before all expected data	General case: - Stream error: stream transfer has been abruptly interrupted. Check that this interruption is management by the application.
1514	Received data while expecting end of stream	General case: - Stream error: downloaded data overflow. Contact your provider.
1515	Stream Invalid Parameter Value Format	General case: - Stream error: Invalid transfered register format. Contact your provider.
1516	Stream Invalid Parameter Type	General case: - Stream error: Invalid transfered register type. Contact your provider.
1517	Error opening file for stream	General case: - Stream error: target file is not accessible. Contact your provider.
1519	Timeout waiting for stream port to be freed	General case: - Cannot connect several times simultaneously from a given application interface.
1550	Could not remove fw update running flag file	General case: - Stream error: The header of the "start firmware download" is incorrect.
1551	Could not remove fw update flag files directory	General case: - Stream error: A problem occurred when writing the downloaded program into flash memory.
1552	Could not create flag files directory	General case: - Firmware management internal error. Download controller firmware again. - If not successful, contact your provider.Contact your provider.
1553	Could not create fw update flag file	General case: - Firmware management internal error. Download controller firmware again. - If not successful, contact your provider.Contact your provider.
1554	Could not remove temporary fw update directory	General case: - Firmware management internal error. Download controller firmware again. - If not successful, contact your provider.Contact your provider.
1555	Could not create temporary fw update directory	General case: - Firmware management internal error. Download controller firmware again. - If not successful, contact your provider.Contact your provider.
1556	Error writing file to flash	General case: - Firmware management internal error. Download controller firmware again. - If not successful, contact your provider.Contact your provider.
1557	Error switching firmwares	General case: - Firmware management internal error. Download controller firmware again. - If not successful, contact your provider.Contact your provider.
1558	Must reboot to take new firmware into account	General case: - Firmware management error: a controller reboot is required.
1559	Attempt to access non existent file	General case: - Firmware management internal error. Download controller firmware again. - If not successful, contact your provider.Contact your provider.
1560	Firmware update failed	General case: - Firmware management internal error. Download controller firmware again. - If not successful, contact your provider.Contact your provider.
1561	Firmware update: failed to create checksum file.	General case: - Firmware management internal error: Download controller firmware again. - If not successful, contact your provider.Contact your provider.
1562	FW update: failed to read file back from flash to verify checksum	General case: - Firmware management internal error: Download controller firmware again. - If not successful, contact your provider.Contact your provider.
1563	FW update: flashed file checksum mismatch	General case: - Firmware management internal error: Download controller firmware again. - If not successful, contact your provider.Contact your provider.
1564	FW update: failed to create temp file	General case: - Firmware management internal error. Download controller firmware again. - If not successful, contact your provider.Contact your provider.
1565	FW update: failed to create directory while setting up SD0 structure	General case: - Please contact your provider.
1566	FW update: failed to format microSD card	General case: - Please contact your provider.

*M64	Description	Solutions
1567	Unable to replace file for update	General case: - Please contact your provider.
1568	FW update: Error changing write protection(s)	General case: - Please contact your provider.
1571	Error getting flash bloc size	General case: - Please contact your provider.
1603	Keep-Alive error	General case: - TCP/IP error: Keep alive timer has expired. Too long time without communication on an open TCP/IP port.
1609	Error accessing IP config element	General case: - TCP/IP error: IP configuration saved data is invalid. Update TCP/IP configuration data from commissioning interface. - if not successful, contact your provider.
1610	Dynamic Invocation server loop exception	General case: - Firmware exception. Please contact your provider.
1620	TCPIP: SetSocketOption error	General case: - Firmware internal error. Please contact your provider.
1621	TCPIP: Bind error	General case: - Firmware internal error. Please contact your provider.
1622	TCPIP: Multicast Start Timeout	General case: - Firmware internal error. Please contact your provider.
1623	TCPIP: Multicast Send message error	General case: - Firmware internal error. Please contact your provider.
1624	TCPIP: IP config error: Refer to sub-Error code	General case: - TCP/IP error: IP configuration cannot be saved in flash. Contact your provider.
1625	TCPIP: IP config saving error: Refer to sub-Error code	General case: - TCP/IP interface error: IP configuration cannot be saved in flash. Contact your provider.
1626	TCPIP: IP config read error	General case: - TCP/IP interface error: IP configuration cannot be read from flash. Update TCP/IP configuration data from commissioning interface. - if not successful, contact your provider.
1627	Application interface error:status change not sent	General case: - TCP/IP interface error: controllers' status data cannot be sent.
1628	TCPIP: Bad startup record	General case: - Firmware exception. Please contact your provider.
1701	Error starting Customer Software Module Function	General case: - Customer Software Module error: the function cannot be started. Check that the function number is correct.
1702	Error stopping Customer Software Module Function	General case: - Customer Software Module error: the function cannot be stopped. Check that this function follows CSM rules provided in User's Manual.
1703	Error stopping Customer Software Module Function. Timeout occurred.	General case: - Customer Software Module error: the function cannot be stopped and timeout is reached. Check that this function follows CSM rules provided in User's Manual.
1704	Customer Software Module not running	General case: - Customer Software Module error: CSM is currently not running. STARTCSM must be called prior to execute CSM functions.
1705	Invalid Customer Software Module number (must be 1)	General case: - Customer Software Module number error: the CSM number must be one.
1710	Failed to autostart Customer Software Module	General case: - Customer Software Module error: autostart function did not succeed starting CSM. Check that the CSM application follows the same structure as provided CSM template.
1711	Failed to load Customer Software Module	General case: - Customer Software Module error: CSM file cannot be loaded. - Download again CSM file and retry. - Check that CSM file size does exceed allowed size.
1712	No runnable CSM	General case: - Either no CSM has been downloaded, or there was but a *NEW=7 or *NEW = 0 command was issued. If no *SAV=0 or *SAV=7 command has been issued since the *NEW, the CSM can be recovered with a *RES=7. If an *SAV has been issued after the *NEW, you need to download the CSM again.

*M64	Description	Solutions
1720	Application interface error: cmd ack not sent	General case: - Command management error: the command sent through TCP/IP cannot be acknowledge. Check your connection.
1721	CSM connection error. Refer to sub-code error	General case: - CSM cannot connect to UA. Refer to UA sub-code error.
1730	File IO Error	General case: - Command management error: internal error. Contact your provider.
1731	Registers files access error	General case: - Registers save handling: some registers cannot be saved. Retry the save command, if no success, contact your provider.
2000	Interpolation error. The number of parameters is incorrect	General case: - Interpolation error: The number of parameters is incorrect.
2001	Interpolation error. Incorrect parameter format	General case: - Interpolation error: Incorrect parameter format.
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.
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.
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.
2005	Interpolation error. IBEGIN command. No node is defined.	General case: - Interpolation error: Ibegin command. No node is defined.
2006	Interpolation error. IBEGIN command. One node is not present.	General case: - Interpolation error: Ibegin command. One node is not present.
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.
2008	Interpolation error. ISET command. One node is not present.	General case: - Interpolation error: Iset command. One node is not present.
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.
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.
2011	Interpolation error. One parameter is out of range.	General case: - Interpolation error: One parameter is out of range.
2012	Interpolation error. The interpolation mode is not active.	General case: - Interpolation error: The interpolation mode is not active.
2014	Interpolation error. Bad parameter in jmp mark action.	General case: - Interpolation error: Bad parameter in jmp mark action.
2015	Interpolation error. Unknown function number in jmp mark action.	General case: - Interpolation error: Unknown function number in jmp mark action.
2016	Interpolation error. The CSM thread is already running	General case: - Interpolation error: Sequence label overflow in jmp mark action.
2019	Interpolation error. The interpolation group number is out of range	General case: - Interpolation error: The interpolation group number is out of range.
2020	Interpolation error. Interpolation group already used	General case: - Interpolation error: Interpolation group already used.
2021	Interpolation error. IBEGIN command: no acknowledge of ITP command	General case: - Interpolation error: IBEGIN command: no acknowledge of ITP command.
2022	Interpolation error. IEND command: no acknowledge of ITP command	General case: - Interpolation error: IEND command: no acknowledge of ITP command.
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.
2024	Interpolation error. Error when all groups do not have the same interrupt refresh rate	General case: - Two interpolated groups in the same interrupt.
2025	Interpolation error. Command counter underflow	General case: - Interpolation error: Command counter underflow.

*M64	Description	Solutions
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.
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).
2049	Interpolation error. Internal error	General case: - Interpolation error: Internal error.
2050	Interpolation error. No interpolation with this UltimET	General case: - Interpolation error: No interpolation with this UltimET.
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.
2060	Interpolation error: axis power off when interpolation active	General case: - Interpolation error: axis power off when interpolation active.
2070	Interpolation error: IBEGIN refresh rate does not match AccurET PLTI (1)	General case: - Interpolation error: The node PLTI does match with interpolation refresh rate.
2071	Interpolation error: IBEGIN refresh rate does not match AccurET PLTI (2)	General case: - Interpolation error: The interpolation update rate is not an integer.
2072	Interpolation error: Numerator or denominator is 0	General case: - Interpolation error: either user's factors' numerator (*K581) or denominator (*K582) is 0. Check that both of these values are different than 0.
3002	Internal error: Refer to sub-Error code	General case: - Command error: Internal management error.
3005	Trace configuration error	General case: - Bad configuration of Traces registers.
3007	Trigger not available	General case: - Trace acquisition error: the selected trace trigger (*K125) is not available. Check this parameter and refer to documentation.
3008	Starting trace with TransnET Sync Reg not 0x0	General case: - Trace acquisition error: multi-axes trace synchronization is not correct on TransnET. Check TransnET connection.
3009	Timeout resetting TransnET traces sync Reg	General case: - Trace acquisition error: multi-axes trace synchronization is cannot be reset on TransnET. Check TransnET configuration.
3030	CHKDSTGRT: inconsistent parameters	General case: - CHKDSTGRT command error: involved parameters' types do not match. Check command syntax.
3031	CHKDSTGRT: Register distance below specified threshold	General case: - CHKDSTGRT command error: registers' distance is below specified threshold. Check that axes target moves.
3500	Axis in mask for synchronized homing not found	General case: - Synchronized homing: Node timeout.
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.
3503	Homing mode not supported for synchronized homing	General case: - Synchronized homing: The current homing type (K40 parameter) is not supported.
3504	Cross Device Feature: axis in error	General case: - Synchronized homing: A node is in error.
3505	Synchronized homing: no index found in either direction	General case: - Synchronized homing: Reference mark has not been detected.
3506	Cross Device Feature: axis in power off	General case: - Synchronized homing: A node is power off.
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.
3510	Synchronized homing: INS returned in error	General case: - Synchronized homing error: a move command returns with error. Check axes error.
3511	Synchronized homing: MVE returned in error	General case: - Synchronized homing error: final move returns with error. Check axes error.

*M64	Description	Solutions
3512	Synchronized homing: command aborted	General case: - Command management error: A command has been aborted. This is a normal behavior due to call of HLT, HLO, HLB command or to an application interface disconnection request.
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
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.
3620	TransnET communication error: time-out waiting for TransnET to disappear after RSH	General case: - TransnET communication Error: Timeout waiting for TransnET to disappear after RSH.
3630	TransnET communication error: time-out waiting for TransnET to re-appear after RSH	General case: - TransnET communication Error: Timeout waiting for TransnET to re-appear after RSH.
3640	TransnET startup error: Inconsistent number of controllers.	General case: - Check if there are two controllers configured with the same DIP switch configuration
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.
4121	An high priority interrupt raised but was not processed	General case: - Internal error: Please contact your provider.
4122	TransnET buffer did not switch	General case: - TransnET internal error: Please contact your provider.
4123	Cyclic RTV handler watchdog error	General case: - Cyclic RTV handler error: watchdog error raises. Check RTV handler execution time.
4124	Cyclic handler watchdog error	General case: - Cyclic handler error: watchdog error raises. Check cyclic handler execution time.
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.
4201	FW is a Recovery Firmware	General case: - Firmware management error: the "Recovery" firmware is currently loaded. This may occur e.g. if FW is corrupted (download failure…): - Try to download again a firmware and reboot the board. - If firmware download is not successful, contact your provider.
4250	Error accessing signature element from flash	General case: - TCP/IP signature elements error: flash signature is either initialized or not accessible. Re-program it through ComET connection chooser. - if not successful, contact your provider.
4500	Invalid IO configuration	General case: - I/O error: Bad digital I/O setting interrupt.
4501	Error state triggered by a DIN	General case: - I/O error: The DIN are in the error state defined by K305.
4502	Error disconnecting IO from pin as local IO	General case: - Local IO error: IO disconnection from pin failed. IO pin is either out of range or used by another function. Check IO configuration.
4503	Error connecting IO to pin as local DIN	General case: - Local IO Error: IO connection to pin failed. IO pin is either out of range or used by another function. Check IO configuration.
4504	Error connecting IO to pin as local DOUT	General case: - Local IO Error: IO connection to pin failed. IO pin is either out of range or used by another function. Check IO configuration.
4505	Some of the IOs requested for direct output are not configured as outputs	General case: - Local IO Error: IO assignment failed. IO pin is used by another function. Check IO configuration.
4506	Some of the DOUTs being set are not available for immediate output	General case: - Local IO Error: IO assignment failed. IO pin is used by another function or not properly configured. Check IO configuration.
4507	Some of the IOs being configured to trigger an error are not configured as inputs	General case: - Local IO Error: IO assignment failed. IO pin is used by another function or not properly configured. Check IO configuration.

*M64	Description	Solutions
4508	Some of the Dins being configured to trigger an error are specified both for level high and low	General case: - Local IO Error: DINs configured to trigger an error are specified both for level high and low. Check IO configuration.
4509	Some of the IOs being configured to react on an error are not pins configured as local outputs	General case: - Local IO Error: DOUTs configured to react on an error are not configured as outputs. Check IO configuration.
4510	Some of the Douts being configured to react on an error are specified both for level high and low	General case: - Local IO Error: Reaction level of DOUTs configured to react on an error are not configured. Check IO configuration.
4511	Some of the IOs being configured to output the TransnET sync signal are not pins configured as local outputs	General case: - Local IO error: IOs configured to output TransnET sync signal are not well configured. Check IO configuration.
4512	Some of the IOs being configured are not available to output the TransnET sync signal	General case: - Local IO error: IOs configured to output TransnET sync signal are not well configured. Check IO configuration.
4513	No pins configured as local DIN	General case: - Local IO error: No pins configured as local DIN. Check IO configuration.
4514	No pins configured as local DOUT	General case: - Local IO error: No pins configured as local DOUT. Check IO configuration.
4516	Douts not available for Ipol	General case: - Local IO error: No DOUTs configured for interpolation marks. Check IO configuration.
4517	Error connecting pin for TransnET signal output	General case: - Local IO error: IOs configured to output TransnET sync signal are not well configured. Check IO configuration.
4518	Multiple of TransnET signal output out of range	General case: - Local IO error: Multiple of TransnET signal output out of range. Adjust parameters.
4600	Same IO number is used on different SPI signals	General case: - SPI error: at least two SPI IO are used for the same signal. Review SPI IO configuration registers.
4601	SPI is already connected	General case: - SPI error: SPI connection is already done. This connection has to be closed before a new connection can be done.
4602	SPI configuration error	General case: - SPI error: one of the SPI IOs is not properly configured. Check IO configuration.
4603	SPI is not connected	General case: - SPI error: a SPI command has been sent when SPI is not configured or properly configured. Check SPI configuration monitoring before to send this command.
4605	SPI command reads/writes to many bytes	General case: - SPI error: too many bytes to transfer in a SPI read/write command. Check command parameters.
4606	SPI pin connection error	General case: - SPI error: IO connection to pin failed. IO pin is either out of range or used by another function. Check IO configuration.
4607	SPI pin disconnection error	General case: - SPI error: IO disconnection from pin failed. IO pin is either out of range or used by another function. Check IO configuration.
4608	SPI not configured	General case: - SPI error: a SPI command has been sent when SPI is not configured or properly configured. Check SPI configuration monitoring before to send this command.
4609	SPI: Error in cyclic task	General case: - SPI error: Error in cyclic task. Check SPI error code. If CRC error, check IO cabling.
4610	SPI: CRC Error	General case: - SPI error: received data do not match with received CRC. Check SPI related IOs connections. - Validate that the other SPI device has the same configuration (CRC, Clock polarity and phase, LSB first…)
4611	SPI: Timeout error stopping SPI in HLx command	General case: - SPI error: Timeout error stopping SPI in HLx command. Please contact your provider.
4700	ESM management: Failed to create ESM	General case: - ESM management error: ESM cannot be created. Contact your provider.

*M64	Description	Solutions
4701	ESM management: ESM Load error. Refer to sub-error code	General case: - ESM management error: ESM cannot be loaded. - Check that the controller firmware is the appropriate one and contains an ESM. - Check *K15 register, which specifies the number of ESM to bloaded. - Otherwise, refer to sub-error code (*M626) and contact your provider.
4702	ESM management: ESM unload error	General case: - ESM management error: ESM cannot be unloaded. Contact your provider.
4703	ESM management: new K15 needs reboot	General case: - ESM management error: reboot the ULTIMET ADVANCED to take into account the new value of K15.
4750	Invalid PTS passwd	General case: - Internal error: Please contact your provider.
4751	PTS test not implemented	General case: - Internal error: Please contact your provider.
4800	FW License file not well formed	General case: - Please contact your provider.
4801	FW License file version not allowed with this FW version	General case: - Please contact your provider.
4802	FW License file controller serial number different than the one in This controller	General case: - Please contact your provider.
4803	FW License file Minimum FW Version is greater than the one in This controller	General case: - Please contact your provider.
4804	No license found for feature	General case: - Please contact your provider.
4805	License file update: Must reboot to take into account.	General case: - Normal behavior. Please reboot to take this license file into account.
4810	FW license file: end of grace period (no license file and features activated)	General case: - Download the license file - Order the missing features that require a license - Disable features that require a license
4811	FW license file: some features cannot be activated because they are not present in the file	General case: - Disable features which are not present in the license file - Order the missing features that require a license

33.2 ESM error codes

*M64 value (Error Code)	Message	Description	Trouble shooting
6100	ESM Core: invalid submodule index in command	(self-explanatory)	Use *M440 to find out the number of submodules in ESM; *ESMGESUBNAME and *M442 to check the index to be addressed.
6160	ESM Core: device absent for RTV setup	Tried to send an ASR/ASW to an axis number not on the TransnET	Check all expected TransnET axes are present
6170	ESM Core: device ASR/ASW failed	The ASR or ASW sent to a device command generated an error on the device.	Contact your provider
6180	ESM Core: no more UA RTV monitoring registers available	The ESM (sub modules assembly) is too RTV greedy	Contact your provider
6190	ESM Core: no more device RTV monitoring register available	Same as above but wrt the drives. Note this only takes into account the Ms handled by the ESM, not any others that would be reserved directly by the application or sequence	Check the device RTV usage and ensure there are some that are free and that those that are needed have been reserved with the *ESMRTVRSVD command.
6200	ESM Core: no more available RTV input channels	The ESM (sub modules assembly) is too RTV greedy	Contact your provider

*M64 value (Error Code)	Message	Description	Trouble shooting
6210	ESM Core: no more available RTV output channels	The ESM (sub modules assembly) is too RTV greedy	Contact your provider
6220	ESM Core: RTV monitoring registers overlap	Some monitoring registers used use the same memory slot.	Check the device RTV monitorings usage.
6230	ESM Core: RTV monitoring register already used	(Self-explanatory)	Check the RTV monitorings usage in the RTV channels requests
6240	ESM Core: no more available RTV read slots	(Self-explanatory) The ESM (sub modules assembly) is too RTV greedy	Contact your provider
6250	ESM Core: no more available RTV write slots	(Self-explanatory) The ESM (sub modules assembly) is too RTV greedy	Contact your provider
6260	ESM Core: attempt to reserve monitoring register that is already used	(Self-explanatory)	Check the RTV monitorings usage in the RTV channels requests

34. 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
Acronyms, 16
ASR, 80
ASW, 79

B
Burst transfer mode, 144

C
CH_BIT_REG32, 39
CHKDISTGRT, 40
CLRWAIT, 45
CLX, 39
Commands
  ASR, 80
  ASW, 79
  CH_BIT_REG32, 39
  CHKDISTGRT, 40
  CLRWAIT, 45
  CLX, 39
  END, 202
  ERR, 49
  ESMRTVRSVD, 67
  ESMSUBACT, 67
  ESMSUBEN, 66
  ESMSUBNAME, 65
  GETDINS, 69
  GETDOUT, 72
  GETMDINS, 69
  GETMDOUT, 72
  HLB, 123
  HLO, 123
  HLT, 123
  IABSCOORDS, 99
  IABSMODE, 98
  IBEGIN, 96
  IBRK, 121
  ICCW, 105
  ICCWR, 106
  ICLRB, 122
  ICONC, 112
  ICONT, 122
  ICW, 105
  ICWR, 106
  IEND, 101
  ILINE, 104
  ILOCK, 122
  IMRES, 135
  IMROT, 134
  IMSHEAR, 134
  IMTRANS, 132
  INCONC, 112
  IND, 52
  IOCFGRST, 70
  IPT, 120
  IPVT, 114
  IPVTUPDATE, 118
  ISET, 95
  ISTP, 121
  ITACC, 109
  ITANSPDMASK, 127
  ITDEC, 109
  ITJRT, 110
  ITSPD, 109
  IUNLOCK, 122
  IWTT, 124
  JMP, 202
  NEW, 42
  RES, 41
  RST, 49
  RSTDOUT, 72
  RSU, 31
  SAV, 41
  SER, 63
  SET_RANGE, 39
  SETDOUT, 72
  SETMDOUT, 73
  SETRTV, 82
  SLT, 51
  SPIABORT, 140
  SPICON, 137
  SPIDIS, 138
  SPIREAD, 139
  SPIRST, 70
  SPIRW, 140
  SPIWRITE, 139
  SSR, 40
  STARTCSM, 201
  STOPCSM, 202
  VER, 63
  WAITCSM, 201
  WARCD, 50
  WBC, 44
  WBS, 44
  WPG, 45
  WPL, 45
  WRST, 50
  WSBC, 44
  WSBS, 44
  WTD, 62
  WTK, 130
  WTM, 43
  WTT, 43
  ZTE, 59
Commands list, 226
Commands syntax, 34
Communication, 17
Concatenated movement, 111
Configuration, 27
Conversion factor, 136
CRC handling, 138
CSM, 150
  From host application, 201
  Special features, 199
  Template, 188
  Troubleshooting, 204
Customer software module, 150
Cyclic transfer mode, 141

D
Data sharing, 76
Development environment, 154
Digital inputs, 68
Digital outputs, 71

E
END, 202
ERR, 49
Errors list, 255
ESM, 64
ESMRTVRSVD, 67
ESMSUBACT, 67
ESMSUBEN, 66
ESMSUBNAME, 65
ETEL Software Module, 64

F
Firmware download, 30

G
G-code based mode, 103
GETDINS, 69
GETDOUT, 72
GETMDINS, 69
GETMDOUT, 72

H
HLB, 123
HLO, 123
HLT, 123

I
IABSCOORDS, 99
IABSMODE, 98
IBEGIN, 96
IBRK, 121
ICCW, 105
ICCWR, 106
ICLRB, 122
ICONC, 112
ICONT, 122
ICW, 105
ICWR, 106
IEND, 101
ILINE, 104
ILOCK, 122
IMRES, 135
IMROT, 134
IMSHEAR, 134
IMTRANS, 132
INCONC, 112
IND, 52
Inputs, 68
Installation, 22
Interpolation movements, 103
IOCFGRST, 70
IPT, 120
IPVT, 114
IPVTUPDATE, 118
ISET, 95
ISTP, 121
ITACC, 109
ITANSPDMASK, 127
ITDEC, 109
ITJRT, 110
ITSPD, 109
IUNLOCK, 122
IWTT, 124

J
JMP, 202

K
K parameters
K15, 64
K16, 201
K33, 70
K40, 52
K120, 57
K121, 57
K122, 57
K123, 57
K124, 57
K125, 57
K126, 57
K127, 58
KD127, 58
KF127, 58
KL127, 58
K128, 58
K129, 58
K140, 48
K141, 47
K144, 124
K166, 50
K171, 71
K177, 46
KL257, 146
K297, 202
K305, 70
K353, 74
K357, 73
K360, 74
K513, 90
K522, 93
K523, 93
K524, 93
K525, 93
K526, 93
K540, 129
K541, 74, 130
K546, 116, 121
K560, 118
K561, 128
K580, 136
K581, 136
K582, 136
K595, 50

K600, 101
K631, 47
K707, 124
K708, 204
K717, 125
K800, 137
K801, 137
K802, 137
K803, 137
K804, 68
K805, 71
K811, 139

M

M monitorings
M50, 68
M63, 46
M64, 255
M66, 50, 254
M70, 63, 65
M71, 63
M72, 63, 65
M73, 63
M84, 65
M85, 63
M87, 17
M90, 46
M95, 190
M97, 190
M100, 201
M120, 58
M121, 58
M122, 58
M123, 58
M124, 58
M125, 58
M126, 58
M127, 58
MD127, 58
MF127, 58
ML127, 58
M128, 58
M129, 58
M131, 58
M134, 58
ML134, 58
M135, 58
M136, 58
M141, 48
M166, 50
M171, 71
M203, 29
M204, 29
M205, 29
M206, 29
M228, 49, 60
ML228, 49, 60
M255, 146
ML255, 145
M256, 146
ML256, 146
ML257, 146
M260, 141
M261, 139
M262, 139
M263, 138
M282, 42
M305, 70
M310 to M373, 80
MD310 to MD341, 80
MF310 to MF373, 80
ML310 to ML341, 80
M439, 65
M440, 64
M441, 65
M442, 65
M443, 79
M444, 66
M445, 66
M447, 80
M448, 79
M449, 79
M512, 90
ML512, 17
M513, 90
M518, 129
M519, 129
M524, 78
M525, 78
ML525, 95
M526, 78
M530, 96
M531, 123
M535, 90
M541, 74
M590, 98
M626, 223
M629, 49
M630, 48
ML630, 47
M640, 103
M641, 103
M642, 103
M643, 103
M644, 103
M645, 103
M646, 103
M647, 103
M649, 109
M650, 109
M651, 109
M652, 110
M658, 94
M659, 94
M660, 103
M661, 103
M662, 103
M663, 103
M664, 103
M665, 103
M666, 103
M667, 103

M669, 109
M670, 109
M671, 109
M672, 110
M680, 127
MF680, 127
M681, 127
M707, 204
M726, 95
M727, 95
M728, 95
M729, 95
M730, 95
M731, 95
M732, 95
M733, 95
M804, 68
M805, 71
M806, 68
M807, 71
M808, 74
M809, 74
Matrix, 131
Monitorings list, 245

N
NEW, 42

O
On-the-fly trajectory change, 117
Outputs, 71

P
Parameters list, 241
Protection signals on digital inputs, 70
Protection signals on digital output, 73
PT mode, 119
PVT mode, 114

R
Real-time channels on TransnET, 76
Registers description, 35
RES, 41
Rotation function, 134
RST, 49
RSTDOUT, 72
RSU, 31
RTV, 76

S
SAV, 41
Scaling function, 133
SER, 63
SET_RANGE, 39
SETDOUT, 72
SETMDOUT, 73
SETRTV, 82
Shearing function, 134
SLT, 51
SPI feature, 136, 137
SPIABORT, 140
SPICON, 137
SPIDIS, 138
SPIREAD, 139
SPIRST, 70
SPIRW, 140
SPIWRITE, 139
SSR, 40
STARTCSM, 201
STOPCSM, 202

T
TCP/IP connections, 22
Translation function, 132
Troubleshooting and known issues, 223

U
UltimET Advanced / Light differences, 220
Units, 92
Units conversion, 266

V
VER, 63

W
WAITCSM, 201
WARCD, 50
Warnings, 50
Warnings list, 254
WBC, 44
WBS, 44
WPG, 45
WPL, 45
WRST, 50
WSBC, 44
WSBS, 44
WTD, 62
WTK, 130
WTM, 43
WTT, 43

X
Xilinx SDK installation, 156

Z
ZTE, 59