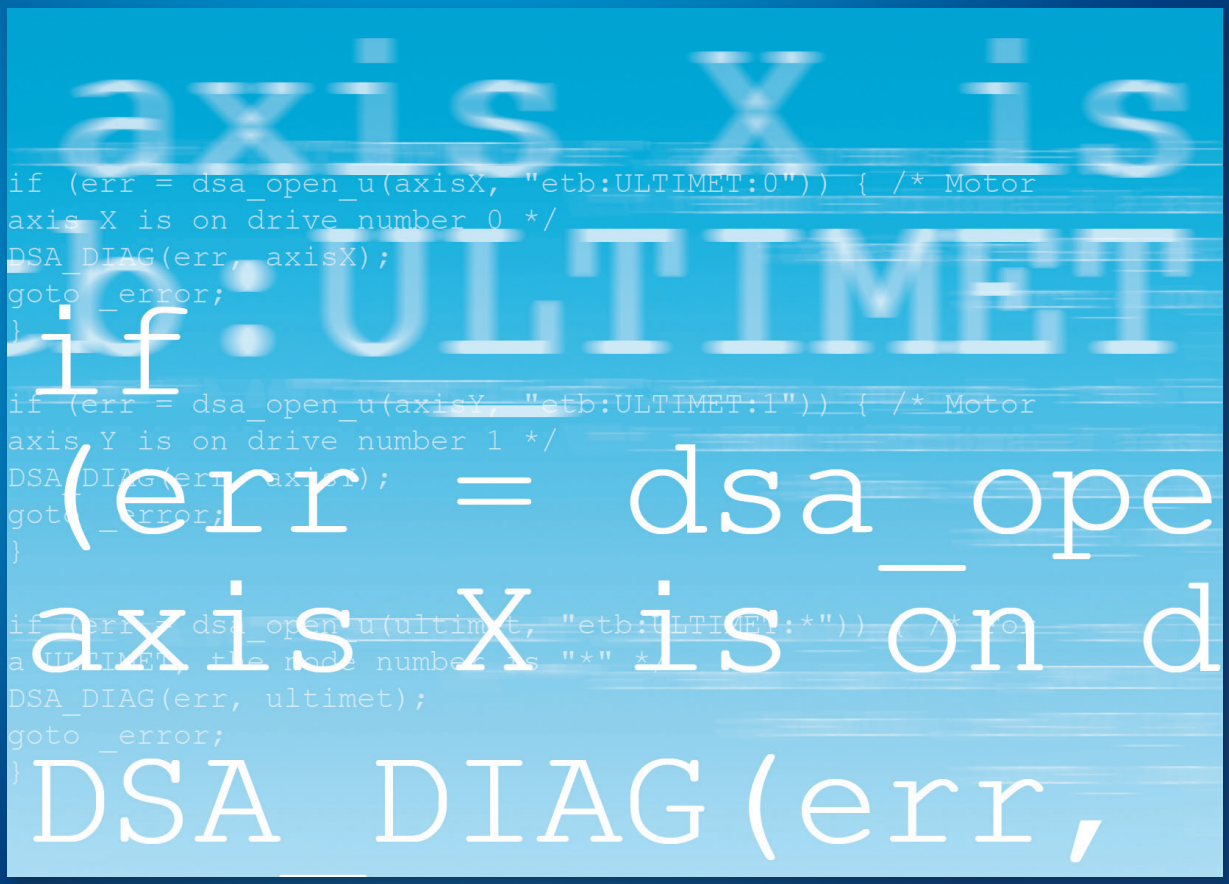


EDI 4
(ETEL Device Interface)

User's Manual

Version M

ETEL

THIS PAGE IS INTENTIONALLY LEFT BLANK

Table of contents

Record of revisions:

Document revisions		
Version	**Date**	**Main modifications**
Ver A	25.04.13	First version
Ver B	05.05.14	Updated version: - New EDI4 package (from EDI 4.10A)
Ver C	17.12.14	Updated version: - New EDI4 package (from EDI 4.12A)
Ver D	27.04.15	Updated version: - New EDI4 package (from EDI 4.13A)
Ver E	30.06.16	Updated version: - New EDI4 package (from EDI 4.15A)
Ver F	06.12.16	Updated version: - New EDI4 package (from EDI 4.16A)
Ver G	25.10.17	Updated version: - New EDI4 package (from EDI 4.17A) - New boolean type (refer to §3.7)
Ver H	29.11.18	Updated version: - New EDI4 package (from EDI 4.20A)
Ver I	29.05.19	Updated version: - New EDI4 package (from EDI 4.21A)
Ver J	05.11.19	Updated version: - New EDI4 package (from EDI 4.22A)
Ver K	18.09.20	Updated version: - New EDI4 package (from EDI 4.23A)
Ver L	25.02.21	Updated version: - New EDI4 package (from EDI 4.24A)
Ver M	24.09.21	Updated version: - New EDI4 package (from EDI 4.25A)

Documentation concerning the EDI4:

- **EDI4** **EDI4 principle & operation**
- **HTML Reference Manual** **File list & data structure in html.bat**

Caution: **As the HTML documentation is generated for each EDI4 release, it is the only up-to-date reference manual.**

All functions present in the "HTML Reference Manual" mentioned under 'Internal use only' are available for the use but are neither documented nor officially supported by ETEL support.

LICENSE EDI4 PACKAGE

Binary Code License

ETEL SA
Binary Code License Agreement

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT") CAREFULLY BEFORE THE OPENING; DOWNLOAD; INSTALLATION OR OTHER MEANS OF USE OF THE SPECIFIED SOFTWARE. BY OPENING DOWNLOAD; INSTALLATION OR OTHER MEANS OF USE, YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCESSING THE SPECIFIED SOFTWARE ELECTRONICALLY, INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT THE END OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL THESE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR, IF THE SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" BUTTON AT THE END OF THIS AGREEMENT.

1. License to use. ETEL grants you a non-exclusive and non-transferable license for the internal use of the specified Software on the number of ETEL products equal to the number of licenses and/or the class of computer hardware for which the corresponding fee has been paid.

2. Restrictions. The specified Software is confidential and copyrighted. Title of this Software and all associated intellectual property rights is retained by ETEL and/or its licensors. Except as specifically authorized in any Supplemental License Terms, you are not entitled to incorporate the Software in whole or in part, in third party product or create derivative works based on the Software, in whole or in part. Furthermore, unless expressly agreed otherwise you may not make copies of the Software, other than a single copy for archival purposes. Unless enforcement is prohibited by applicable law, you may not modify, decompile, and reverse engineer this Software.
This Software is not designed or licensed for use in on-line control of aircraft, air traffic, aircraft navigation or aircraft communications; or in the design, construction, operation or maintenance of any nuclear facility. Licensee represents and warrants that it will not use or redistribute the Software for such purposes. You may not publish or provide the results of any benchmark or comparison tests run on this Software to any third party without the prior written consent of ETEL. No right, title or interest in or to any trademark, service mark, logo or trade name of ETEL or its licensors is granted under this Agreement.

3. Limited warranty. ETEL warrants to you that for a period of ninety (90) days from the date of purchase, as evidenced by a copy of the receipt, the media on which Software is furnished (if any) will be free of defects in materials and workmanship under normal use. Except for the foregoing, Software is provided "AS IS". Your exclusive remedy and ETEL's entire liability under this limited warranty will be at ETEL's option to replace Software media or refund the fee paid for Software.

4. This Software is provided "AS IS" without a warranty or representations of any kind. All express or implied conditions, representations and warranties, including any implied warranty of merchantability, fitness for a particular purpose or non-infringement are hereby excluded. ETEL and its licensors shall not be liable for any damages suffered by licensee as a result of using, modifying or distributing the Software or its derivatives. In no event will ETEL or its licensors be liable for any lost revenue, profit or data, or for direct, indirect, special, consequential, incidental or punitive damages, however caused and regardless of the theory of liability, arising out of the use of or inability to use Software, even if ETEL has been advised of the possibility of such damages. The entire risk arising out of use, performance or non-performance of the Software remains with the licensee. If ETEL should nevertheless be found liable, whether directly or indirectly, for any loss, damage or injury arising under this agreement or otherwise, regardless of cause or origin, on any basis whatsoever, its total maximum liability is limited to CHF 100'000 which will be the complete and exclusive remedy against ETEL.

5. Termination. This Agreement is effective until terminated. You may terminate this Agreement at any time by destroying all copies of Software. This Agreement will terminate immediately without notice from ETEL if you fail to comply with any provision of this Agreement. Upon Termination, you must destroy all copies of Software.

6. Export Regulations. All Software and technical data delivered under this Agreement are subject to Swiss export control laws and may be subject to export or import regulations in other countries. You agree to comply strictly with all such laws and regulations and acknowledge that you have the responsibility to obtain such licenses to export, re-export, or import as may be required after delivery to you.

7. Governing Law. Any action related to this Agreement will be governed by Neuchatel law and controlling Swiss federal law. No choice of law rules of any jurisdiction will apply.

8. Severability. If any provision of this Agreement is held to be unenforceable, This Agreement will remain in effect with the provision omitted, unless omission would frustrate the intent of the parties, in which case this Agreement will immediately terminate.

9. Integration. This Agreement is the entire agreement between you and ETEL relating to its subject matter. It supersedes all prior or contemporaneous oral or written communications, proposals, representations and warranties and prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter during the term of this Agreement. No modification of this Agreement will be binding, unless in writing and signed by an authorized representative of each party.

10. Alpha, beta, and work versions. For software products designated as alpha, beta or work versions, the following particular provisions shall apply: Software products designated as alpha, beta or work versions are provided to the user free of charge, expressly as unfinished software versions still under development, and exclusively for trial and evaluation purposes. Any use for other purposes, in particular for productive purposes in or as part of production systems is expressly forbidden. The user is obliged to make sure that, by using alpha, beta or work versions, no systems will be affected that are relevant to the safety of persons or goods in any form whatsoever. ETEL expressly points out that the software provided does not contain all software functions intended for the final version. Accordingly, the user may not claim that the requested task can be solved with the alpha, beta or work versions of the software. Furthermore, it is not guaranteed that functionalities contained in the alpha, beta or work versions will also be part of a later final version. Accordingly, the user is aware of the fact that software alpha, beta or work versions are provided without any warranty of quality, without any intended use whatsoever, and without any promises related to safety / reliability. Therefore, any information on the software shall not be binding, even if indicated otherwise. In addition, any suitability for common use or usual quality are ruled out. This means that no corresponding warranty or guarantee for this software can be granted. This further means that any liability by ETEL to this effect in case of damage will already be excluded due to the lack of any breach of a corresponding obligation. The aforementioned software is used exclusively at the user's own risk. If ETEL is held liable by a third party because the third party has suffered damage, the user shall release ETEL from all justified claims resulting from this liability.

For inquiries please contact: ETEL SA, Zone Industrielle,
2112 Môtiers, Switzerland

Supplemental license terms

These supplemental terms ("Supplement") add to the terms of the Binary Code License Agreement ("Agreement"). Capitalized terms not defined herein shall have the same meanings ascribed to them in the Agreement.

The Supplement terms shall supersede any inconsistent or conflicting terms in the Agreement.

1. License to Distribute. You are granted a royalty-free right to reproduce and distribute the Software provided that you: (i) distribute the complete Software and unmodified, only as part of, and for the purpose of tuning, upgrading, debugging... your application into which some ETEL controllers are incorporated; (ii) do not distribute additional software intended to replace any component(s) of this Software; (iii) do not remove or alter any proprietary legends or notices contained in the Software; (iv) only distribute the Software subject to a license agreement that protects ETEL's interests consistent with the terms contained herein; and(v) agree to indemnify, hold harmless, and defend ETEL and its licensors from and against any claims or lawsuits, including attorneys' fees, that arise or result from the use or distribution of this Software.

2. High Risk Activities. Notwithstanding Section 2, with respect to high risk activities, the following language shall apply: the Software is not designed or intended for use in on-line control of aircraft, air traffic, aircraft navigation or aircraft communications; or in the design, construction, operation or maintenance of any nuclear facility. ETEL disclaims any express or implied warranty of fitness for such uses.

THIRD-PARTY LICENSES

- **7ZIP license**

EDI Windows version uses 7ZIP functionality, especially for System Configuration Management.
7z.exe and 7z.dll are piece of binary code, provided with EDI package. The source code can be found here: http://www.7-zip.org. These piece of program are subject to following licensing: 7-Zip

License for use and distribution

7-Zip Copyright (C) 1999-2011 Igor Pavlov.

Licenses for files are:
1) 7z.dll: GNU LGPL + unRAR restriction
2) All other files: GNU LGPL

The GNU LGPL + unRAR restriction means that you must follow both GNU LGPL rules and unRAR restriction rules.

Note: You can use 7-Zip on any computer, including a computer in a commercial organization. You don't need to register or pay for 7-Zip.

GNU LGPL information: This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.
You can receive a copy of the GNU Lesser General Public License from http://www.gnu.org/

unRAR restriction: The decompression engine for RAR archives was developed using source code of unRAR program. All copyrights to original unRAR code are owned by Alexander Roshal. The license for original unRAR code has the following restriction:
The unRAR sources cannot be used to re-create the RAR compression algorithm, which is proprietary. Distribution of modified unRAR sources in separate form or as a part of other software is permitted, provided that it is clearly stated in the documentation and source comments that the code may not be used to develop a RAR (WinRAR) compatible archiver.

Igor Pavlov

- **LWIP license**

EDI FreeRTOS version uses LWIP TCP/IP stack, delivered under BSD license:

Copyright (c) 2001-2004 Swedish Institute of Computer Science. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Author: Adam Dunkels adam@sics.se

1 Introduction

1.1 Acronyms

Abbreviation	Definition
API	Application Program Interface
DLL	Dynamic Link Library
EDI	ETEL Device Interface
ISO	International Organization for Standardization
URL	Uniform Resource Locator
CSM	Customer Software Module (ULTIMET ADVANCED embedded user application based on EDI)
ESM	ETEL Software Module (ULTIMET ADVANCED module integrated in firmware)
SHM	Shared memory (communication between Customer Software Module and ULTIMET ADVANCED firmware)

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

1.2 Glossary

The following terms are constantly used in this manual. It is essential to know these definitions before reading this manual.

AccurET ETEL position controller.

AccurET-family Devices belonging to the AccurET family and communicating through TransnET (AccurET and UltimET Light or ULTIMET ADVANCED).

API Application Programming Interface: a set of software functions to have access to a system.

Device Can be a position controller (i.e. AccurET), a motion controller (i.e. UltimET Light or ULTIMET ADVANCED).

DLL «Dynamic Link Library». A .dll file contains one or several functions compiled, linked, and stored separately from the processes using them. The operating system maps the DLLs into the address space of the process when this process starts up or as they are called, while it is running. The process then executes the functions in the DLL.

Drive Equivalent to an AccurET position controller.

EDI4.xx Set of libraries supporting AccurET-family products, **but not the DSC-family and DSB-families.**

Increments ETEL's devices use a large number of physical quantities (position, speed, acceleration, force, time, electric current,...). These quantities are represented by 32 or 64-bit integers or floats, with units specific to the devices (UPI, USI,...). These units are called 'increments', hence the representation by increments' term used to represent the physical quantities.

IRQ Interrupt Request.

ISO units International system of units used to express the physical quantities. The basic units are m(meter), s (second), kg (kilogram), A (Ampere) and K (Kelvin). There are also their derivatives: m/s, m/s^2, m/s^3, N (Newton) = kg * m/s^2, W (Watt) = N * m/s, V (Volt) = W/A,...

Remark: For EDI, the temperature is always given in °C (Celsius) and the angular positions in turns.

ISR Interrupt Service Routine.

Master	Refers to an UltimET motion controller in the context of the TransnET communication bus.
Motion controller	ETEL device that is able to control several position controllers (e.g. UltimET Light or ULTIMET ADVANCED motion controllers).
OS	Operating System.
Object	Refers to the definition widely used in the object-oriented programming. In C language, an object is nothing more than a structure containing a series of information (a classical example of a C object is the FILE structure of the standard library).
Position controller	ETEL device capable of controlling the position of an axis or axes (e.g. AccurET position controllers).
RTV	Real-Time Value.
Slave	Refers to an AccurET position controller in the context of the TransnET communication bus.
TransnET	Very high speed proprietary communication bus used to exchange information between the AccurETs and between AccurETs and UltimET.
UltimET	Generic term referring to UltimET Light or ULTIMET ADVANCED motion controllers.
UltimET Light	Refers to UltimET Light PCI, PCIe or TCP-IP products motions controllers.
ULTIMET ADVANCED	Refers to ULTIMET ADVANCED motion controller.

1.3 Presentation of this manual

1.3.1 Intended readers

It is assumed the user is familiar with software programming, in particular the C language. Software design and architecture aspects must be mastered by the developer intending to use this DLL interface. In particular integrating a DLL into a software project should not be an issue.

Also the user must be familiar with ETEL position controllers and their usage as described in the corresponding "Operation and Software Manual". If the application requires a motion controller, the developer must refer to the corresponding "User's Manual" to understand how to use it.

1.3.2 Contents

This document presents all the libraries of the EDI package with emphasis on the DSA library. This library is the only one that is necessary to program the controllers from a PC. The other libraries are mainly used by the DSA library itself.

The next chapter describes the general package including its structure, supported operating systems, communication buses and file description.

The chapters that follow are ordered in pretty much the same way as one would go about developing an application using ETEL controllers. They detail how to use the library by referring to an example that is the same throughout the different chapters.

Throughout the example, the basic and most common operations will be reviewed. The complete list of operations provided by the package can be found in the **"HTML Reference Manual"** organized by feature.

The appendices include:
- Table containing for each controller command the corresponding DSA function, when it exists;
- Similar table containing for each device register the corresponding access functions, when they exist;
- Description of the error output;
- Description of the examples provided, other than the one analyzed throughout this manual;
- Description of the changes required to upgrade an application from EDI3xx to EDI4.xx.

2 Package features

2.1 Presentation

The ETEL Device Interface (EDI) is a set of libraries which enable the communication with ETEL position and motion controllers to access their functionalities. ETEL began to develop EDI in 1997. Four major versions have been developed, but the table below only includes information about the two most recent ones (EDI3.xx and EDI4.xx).

EDI version	First release	DSC family	AccurET family	Development environment	EDI documentation
EDI3.xx	3.00A (17.03.2009)	✓	✓	Visual Studio 2005	From EDI-user-c.pdf
EDI4.xx	4.01A (03.06.2013)		✓	Visual Studio 2010 until 4.16A Visual Studio 2015 until 4.22A Visual Studio 2019 from 4.23A	From EDI4-User-VerA.pdf

Remark: Refer to §15.2 for more information about the changes between EDI3.xx and EDI4.xx.

These libraries constitute an API (Application Program Interface) allowing the user to:
- Manage the communication with the controllers through the many existing communication buses, e.g. PCI/PCIe, TCP/IP, USB and Shared Memory (SHM).
- Access all the functionalities of ETEL controllers through an homogeneous interface, whatever the product and firmware version.
- Manage the conversion of the products' internal units (increments) in order to enable the user to work with conventional units (ISO).

These libraries are totally developed in C and can be used from other common programming languages, such as C++, and C#. There is a DLL to be used from C/C++ applications and a class library for C# .NET framework version 4.8.

Most of the applications call the DSA high level library, which in turn calls the other libraries of the EDI package. This library is able to manage the latest position and motion controllers.

There is a `readme.doc` file that gives the installation and compilation guidelines for the supported Operating Systems, as well as the list of files that are used and where they can be found.

2.2 Supported Operating Systems

The EDI set of libraries includes support for the following operating systems:
- Windows 10 Enterprise and
- RTX64 4.0 (64-bit).

Remark: EDI may run on other Windows and RTX operating system versions. However, the qualification of the EDI libraries has been realized under the conditions listed in the qualification environment table presented below. ETEL cannot guarantee full technical support for issues occurring under different operating system conditions.
ETEL does not support operating systems that are no longer supported by their respective vendors.
The PCIe interface of the ULTIMET ADVANCED motion controller is not supported on the RTX operating system.

EDI Qualification Environment		
Windows and RTX 64-bit	Operating System	RTX 64 4.0 (2-core) / Windows 10 Entreprise LTSC 64-bit (2-core)
	Processor Type	Intel® Core™ i7-7700 CPU @ 3.60 GHz
	System memory (RAM)	16 GB
	EDI development environment	Visual Studio 2019
	Application type	64-bit application

Special RTX recommendation: from EDI-4.10A, it is advised to configure RTX to request its memory from local memory pool. Refer to §14 for RTX configuration.

2.3 DLL (Dynamic Link Library)

The EDI package for Windows is a set of dynamic link libraries commonly known as DLL.

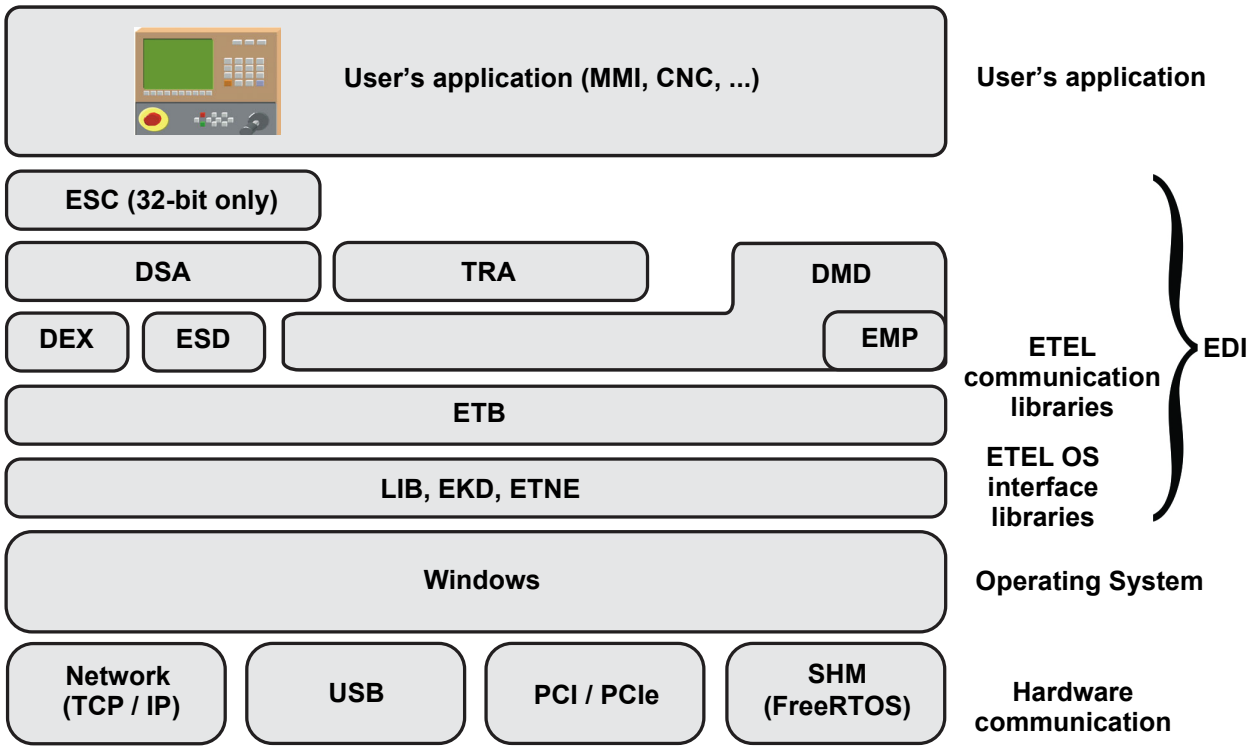
This kind of library is a standard for Windows and all programmers are used to deal with it. Actually, all the Windows' libraries are in this format and all the compilers available for Windows use this kind of libraries. So, DLLs are characterized by:

- Being a Windows standard;
- Their format is known to all compilers;
- They are dynamically linked, which allows them to be modified or updated without requiring recompilation of the applications using them.

To use the EDI package with Windows it is important to correctly understand the functioning of a DLL. Please refer to the manuals of your compiler to know in detail how the DLLs work and how the compiler is able to manage them. Be careful to understand where the DLLs have to be stored and where the application accesses them.

2.4 Package structure

The EDI package is in fact a collection of several libraries. However, the API for programming ETEL controllers is provided entirely by the DSA library. The other libraries are only used by the DSA top-level library and the developer does not need to access them directly.



For reference, here follows an explanation of the functionality provided by each one of these libraries:

ESC	• Allows the compilation as well as the download and upload of a Sequence of the AccurET family. This library makes uses of a help library named assert40c.dll and is only available for 32-bit Operating Systems.
DSA	• Manages a single device, device group and interpolation device group objects; • Generic functions for using all the functionality of the ETEL devices; • Performs conversions between the device internal units (increments) and ISO units; • Transparent to communication configuration; • Manages devices on several communication buses at the same time; • Specific functions to set and get the main device registers; • Specific functions to execute most commands used by the devices; • Asynchronous and synchronous functions.
TRA	• Manages the translation between the ETEL language (terminal commands) and the ETCOM records internally used by the controllers.

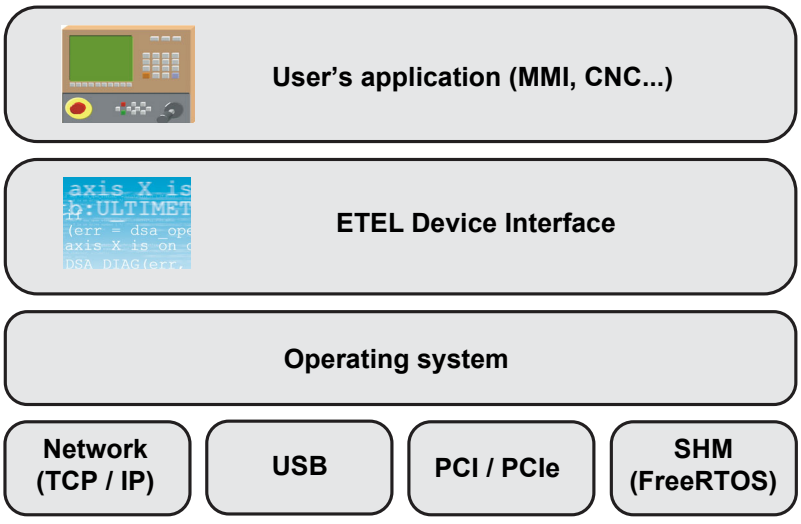
DMD	• Manages meta-information objects; • Stores min / max and default values for each register; • Stores the list of spaces / indexes and subindexes available; • Translates error codes / status / commands and aliases; • Directly generated from the position controller database; • Stores information for each device and firmware version.
DEX	• Allows exporting and importing Time-based (Scope), Frequency-based (Identification), Mapping and Scaling data to or from a file.
ESD	• Allows downloading a pre-compiled Sequence file.
EMP	• Parses XML metadata files.
ETB	• Manages the communication buses and logical ports; • Handles the PCI/PCIe, USB and TCP/IP transparently; • Handles the normal and boot communications; • Handles the status update (refer to §7) • Multiple logical port message queuing; • Registering of event handlers (callback).
LIB and EKD	• Help libraries for hardware and operating system specific accesses.
ETNE	• Allows communication to the ETEL ETND deamon which is a process acting as a TCP/IP server. By running this process, it is possible for another network user to connect to ETEL controllers' local hardware.

2.5 Main characteristics

The main characteristics of the software are:
- Implemented in ANSI C;
- Can be used from C, C++ and C# (refer to §2.1 for further details);
- Common source files for multiple platforms and Operating Systems;
- C++ wrapper classes included in header files;
- Thread-safe implementation;
- Object-oriented architecture.

2.6 Supported communication buses

2.6.1 Description



At the moment, the EDI package supports the following communication buses:

Communication bus	ETEL product
PCI / PCIe	UltimET Light PCI, UltimET Light PCIe and ULTIMET ADVANCED PCIe
TCP/IP	AccurET, UltimET Light TCP/IP and ULTIMET ADVANCED TCP/IP
USB	AccurET
SHM	ULTIMET ADVANCED (Customer Software Module)

UltimET Light PCIe is supported since EDI 4.11A. This board is MSI (Message Signal Interrupt) capable. To

use the MSI capability of this board, the UltimET PCI driver 11.1.0 or higher must be installed. If another driver is used, the MSI capability will not be enabled. Caution must be taken when installing the UltimET 11.1.0 driver.

ULTIMET ADVANCED PCIe is supported since EDI 4.22A. The same driver as the UltimET Light PCI/PCIe can be used, but it must be the driver delivered with the EDI 4.22 package or higher. Please refer to the UltimET PCI/PCIe Technical Note available in the documentation folder of the EDI and ComET packages or the corresponding "UltimET motion controller User's Manual".

Remark: The PCIe interface of the ULTIMET ADVANCED motion controller is not supported on the RTX operating system.

With the package comes an ETEL network process called «ETND», which is a daemon running on a remote machine. This distant machine is connected to the controller system via any of the other aforementioned buses. The application can connect to this remote «ETND» on a distant machine via a TCP/IP and have access to the remotely connected controllers.

Restrictions:
Only one process can be connected to a specific bus at the same time. The ETND/ETNE mechanism if the user wants to have several processes connected to the same hardware through the same bus. Please refer to the ETND-ETNE Technical Note for further information.

2.7 Versions numbering

All the ETEL software programs are identified with a version numbering as follow:

<digit 1>.<digit 2><digit 3><alphabetical character 1>

For example, a version number can be: 1.00A, 2.00D, 3.15B, etc. Each EDI package and library have their own version number. There is no direct link between the package and the individual libraries' version numbers. For the libraries, the first digit (<digit1>), representing the major version index of the version number, is present followed by a 0 in the file name of this library. To know the exact version of a library, open Windows Explorer and right click on the library file to select 'Properties'. To get the general EDI Package version, the DSA library provides the function `dsa_get_edi_version()`.

2.8 Files for Windows

For each library, there are several files necessary for the compilation and the execution of the code. To run EDI, the following files are used:

dsa40.h	Header file necessary during the compilation
dsa40c.lib	Static library necessary during linking
dsa40c.dll, tra40c.dll, etb40c.dll, dmd40c.dll, emp40cd.dll, esd40c.dll, ekd40c.dll, (or ekd40_32c.dll if you are running a 32 bits application on Windows 64 bits OS), lib40c.dll, dex40c.dll	Dynamic link library necessary during execution
esc40c.dll, assert40c.dll	Dynamic link library necessary to compile an ETEL sequence (32-bit only)
etne40c.dll	Dynamic link library necessary to allow TCP/IP connection through ETND daemon
FTBUSUI.dll, FTD2XX.dll	Dynamic link library necessary to connect through USB
wdapi910.dll, wdapi1021.dll (or wdapi1021_32.dll if you are running a 32-bit application on Windows 64-bit OS) wdapi1110.dll (or wdapi1110_32.dll if you are running a 32-bit application on Windows 64-bit OS)	Dynamic link library necessary to connect to UltimET PCI/PCIe

The debug versions, necessary during the development, are added to these files:

dsa40cd.lib	Static library, debug version
dsa40cd.dll, tra40cd.dll, etb40cd.dll, dmd40cd.dll, emp40cd.dll, esd40cd.dll, ekd40cd.dll, (or ekd40_32c.dll if you are running a 32 bits application on Windows 64 bits OS), lib40cd.dll, dex40cd.dll	Dynamic link library, debug version
esc40cd.dll, assert40cd.dll	Dynamic link library necessary to compile an ETEL sequence (32-bit only)
etne40cd.dll	Dynamic link library necessary to allow TCP/IP connection through ETND daemon

wdapi910.dll, wdapi1021.dll (or wdapi1021_32.dll if you are running a 32-bit application on Windows 64 bits OS) wdapi1110.dll (or wdapi1110_32.dll if you are running a 32-bit application on Windows 64 bits OS)	Dynamic link library necessary to connect to UltimET PCI/PCIe

The files contain the name of the library followed by a number which represents the first digit of the version number. One or several letters enable the differentiation of the implementation types.

The .lib files are simple interfaces between the application and the DLL. Their size is then relatively small because the code of the functions is stored in the DLL file. These files are given in the COFF format for Visual Studio 2019 C++.

For the libraries other than DSA, the files are more or less the same. These libraries are internally used by the DSA library or for special applications.

Remark: ETEL does not provide any support concerning the use of other libraries such as TRA, DMD, ETB, ESC, ETNE, EKD, LIB, ESD, EMP and DEX.

3 General guidelines for programming with the EDI package

3.1 Object oriented

The DSA library as all the other libraries of the EDI package, is totally implemented in C and follows the object-oriented philosophy. The work is done with objects represented by C structures. Hence, an object corresponds to each device, and all the functions concerning this device receive a pointer to this object (this structure) as parameter. The content of the structure is not visible to the user.

This is a similar approach to the functions of the standard C library which give access to files. Here also, an object (FILE structure) is created for each file the user wants to manage and all the functions doing an operation on a file receive as parameter the pointer to this object. It is the case of fprintf(), which receives as first parameter the pointer on the FILE structure.

To have access to a position controller, a pointer on the `DSA_DRIVE` object must be first created as follows:

```
DSA_DRIVE *drive1 = NULL;
```

It is important to assign this pointer to the NULL value. To avoid wrong manipulations of the pointer, the library refuses to create an object if the pointer is not NULL.

Once the pointer declared, the object must be created and the pointer affected. This is done as follows:

```
err = dsa_create_drive(&drive1);
```

As mentioned above, most of the DSA functions return an integer which represents the error code. To detect a possible error, the creation of a drive object can be done as follows:

```
DSA_DRIVE *drive1 = NULL;
int err;

err = dsa_create_drive(&drive1);
if (err != 0) {
  printf("cannot create drive object\n");
  exit(1);
}
```

The creation of an object, necessary for the communication with the UltimET, is done in a similar way:

```
DSA_MASTER *UltimET = NULL;
int err;

err = dsa_create_master(&UltimET);
if (err != 0) {
  printf("cannot create UltimET object\n");
  exit(1);
}
```

3.2 Groups

In the previous section, the first parameter of each DSA library function represents the device on which the user wants to do the operation. In practice, this parameter is a pointer to an object which includes all the information of the device in question.

However, the user often needs to send a command or do an operation on several controllers at the same time. This is also the case when the user wants to start synchronized movements. For that purpose, the DSA library has the possibility to create groups of devices. The user can then perform operations on these groups instead of performing them on an individual device.

The DSA library enables the definition of the following groups:

Group	Type	Description	Product
Device group	DSA_DEVICE_GROUP	Group of several position or motion controllers of the same family.	AccurET, UltimET
Drive group	DSA_DRIVE_GROUP	Group of several position controllers of the same family.	AccurET
UltimET group	DSA_MASTER_GROUP	Group of several motion controllers of the same family.	UltimET
Interpolation group	DSA_IPOL_GROUP	Group of several controllers with its motion controller which generate interpolated movements with the controllers of the same family.	AccurET, UltimET

To create a group, the process is similar to the one used to create a single device. A pointer to the object corresponding to the desired group must be first created:

```
DSA_DRIVE_GROUP *group1 = NULL;
```

Like for the drive and master objects, the pointer must be assigned to NULL before creating the group object. Then, the following function must be called:

```
err = dsa_create_drive_group(&group1, 2);
```

Compared to the creation of a drive, the creation of a group requires an extra parameter which is the number of devices that the user wants to assign to the group. It is in a way the size of the group which is equal to 2 in the above example.

Once the group created, the devices belonging to it must be assigned as follows:

```
err = dsa_add_group_item(group1, drive1);
err = dsa_add_group_item(group1, drive2);
```

To do so, both `drive1` and `drive2` objects must have been created before.
Instead of allocating a device, it is possible to assign another group. For example, the following group can be created:

```
dsa_create_drive_group(&group2, 2);
dsa_add_group_item(group2, drive3);
dsa_add_group_item(group2, group1);
```

In the above example, a group of position controllers (`DSA_DRIVE_GROUP`) has been created. Therefore, only `DSA_DRIVE, DSA_DRIVE_GROUP`, or the `DSA_IPOL_GROUP` objects can be assigned to it. If the user wants to include the UltimET in a group, the `DSA_DEVICE_GROUP` group type must be created.
A group of position controllers can include position controllers which are not on the same communication bus. On the other hand, in a `DSA_IPOL_GROUP`, the position controllers are only accessible via the same UltimET which means that they must be connected to the same TransnET (refer to the "UltimET Light or ULTIMET ADVANCED User's Manual" for more details about the interpolation group).
In most cases, a group can be used exactly like a device. It is then possible to use a group as the first parameter of the DSA functions as in the following example, where the «power on» command is sent to all position controllers of group 1:

```
err = dsa_power_on_s(group1, ...);
```

A `DSA_IPOL_GROUP` can also be used where a `DSA_DEVICE_GROUP` or `DSA_DRIVE_GROUP` can, as follows:

```
DSA_IPOL_GROUP *igroup = NULL;
dsa_create_ipol_group(&igroup, 2);
dsa_add_group_item(igroup, drive3);
dsa_add_group_item(igroup, group1);
err = dsa_power_on_s(igroup, ...); /*send power on to drive 1, 2, and 3*/
```

But, a `DSA_DEVICE_GROUP` cannot be used everywhere as a `DSA_IPOL_GROUP` can. The function starting the interpolation mode only accepts a `DSA_IPOL_GROUP` as parameter:

```
dsa_ipol_begin_s(igroup, ...); /*but not dsa_ipol_begin_s(group1,...)*/
```

Also, it is not possible to read a register on a group and so the following example is invalid:

```
err = dsa_get_register_s(group1,...);    /* WRONG */
```

See §3.3 for more details about the hierarchy of the different objects and which type of object is accepted by each DSA library function.

3.2.1 Gantry

ETEL provides several ways to define a GANTRY:

- **Gantry level 2:**

This is the most convenient way to use a gantry (refer to the "AccurET Operation & Software Manual"). In this mode, the gantry management is done by the AccurET itself. Of course, the 2 axes of the gantry must be the two axes of the same AccurET controller.
In this level, the user can define and open communication only on a single `DSA_DRIVE`, which must be the gantry master. Of course, it is possible to open the communication on the slave, if the user wants to monitor some registers of the slave.

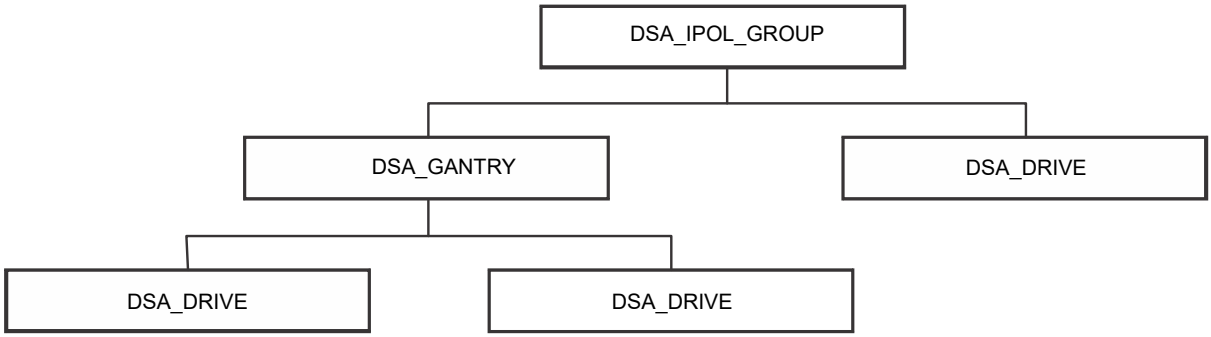
- **Gantry level 1:**

In this level, the user has to define and open the communication with the 2 axes of the gantry. Moreover, the user should define an EDI `DSA_GANTRY` object, which is, in fact, a group of these 2 axes. The commands should be sent using this `DSA_GANTRY` group, especially for the "Homing" command. In this gantry level, the homing procedure is handled by the UltimET. Inside EDI, the `dsa_start_homing_s/a` function will redirect the homing to UltimET if the device parameter is a `DSA_GANTRY`.

- **Group of group**

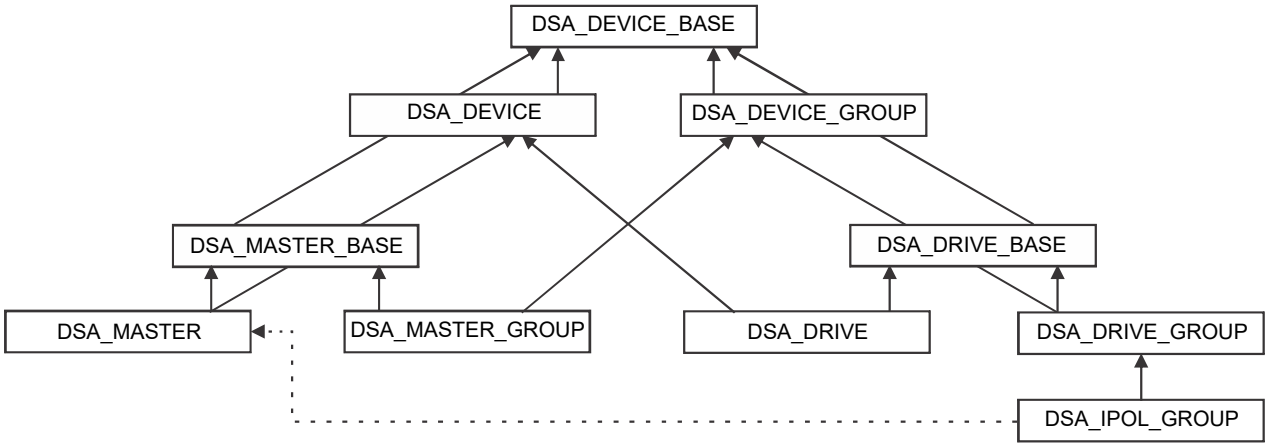
EDI can handle a group of groups as long as they are on the same bus. This is especially useful to make interpolation between a single axis and the 2 axes of a gantry in level 1. In this case, the user must define:

- 3 `DSA_DRIVE` (the single axis and the 2 axes of the gantry);
- 1 `DSA_GANTRY` (which is a group of 2 axes), filled with the 2 axes of the gantry;
- 1 `DSA_IPOL_GROUP` filled with the `DSA_GANTRY` object and the single axis.



The `DSA_IPOL_GROUP` can then be used on all EDI interpolation functions.

3.3 Inheritance



As seen previously, the DSA library functions have always, as first parameter, an object which represents a device or group of devices. Nevertheless, it is not possible to do all operations on any object.

For example, the power of a position controller can be switched on and off, but it is not possible to do it with an UltimET. The `dsa_power_on_s()` will hence accept the objects of `DSA_DRIVE` and `DSA_DRIVE_GROUP` type, but not the `DSA_MASTER` type.

Similarly, reading of a register can only be done on one device at the same time. The `dsa_get_register_...()` function will then accept the objects of the type `DSA_DRIVE` and `DSA_MASTER` but not the `DSA_DRIVE_GROUP` type.

The objects of the DSA library are organized in a hierarchy which defines the relations of heritage commonly used in oriented-object programming. This hierarchy is illustrated above.

On the above figure, the `DSA_DRIVE` object inherits the `DSA_DEVICE` object which means that the `DSA_DRIVE` «is» a `DSA_DEVICE` or in other words that the `DSA_DRIVE` can be used instead of a `DSA_DEVICE`. If a DSA function accepts a `DSA_DEVICE` as parameter, it will automatically accept a `DSA_DRIVE` too, which is the case for the `dsa_get_register_...()`.

From these relations of heritage and the prototype of the function, it is easy to know which objects are supported.

3.4 Functions

There are two types of functions provided in the EDI package: those that return when the execution of the feature it implements is finished (these are called synchronous functions) and those that return immediately without waiting for the task they have started to terminate (these are called asynchronous functions). Usually, for each function implementing a specific feature, the two type (synchronous and asynchronous) are available. They can be easily identified by the appending letters on their respective names:

- **_s** for synchronous functions
- **_a** for asynchronous functions

For example, to switch the power on, one can either use the `dsa_power_on_`**`s`**`(...)` or the `dsa_power_on_`**`a`**`(...)` function, depending on whether if it is intended that the execution flow in the application waits for the end of the «power on» or not, in order to do other things in the meantime.

The first parameter of each function is always the `DSA_DRIVE` or `DSA_MASTER` object which represents the device on which the operation will be executed.

3.4.1 Synchronous functions and timeouts

These functions are called by the user and their execution ends once the respective operation is finished.

For example, when the `dsa_get_register_..._s()` function is called, it asks the device for the value of a register and waits for the reply. Once the device has returned the value of the corresponding register, the function exits.

The synchronous functions put on hold the execution of the program until its operation is concluded. It stops the user from doing another things before the current one is finished. A way of bypassing this is to use multiple threads or resorting to asynchronous functions (see §3.4.2).

In the DSA library, the synchronous functions are identified by a '**_s**' affix on their name and by a 'timeout' parameter which is always the last one. The timeout is the maximum time allowed for the operation to finish. This time is always expressed in milliseconds. To use the synchronous version of the «power on» function, one would write:

```
if (err = dsa_power_on_s(drive, 10000)){
   DSA_EXT_DIAG(err, drive);
   goto _error;
}
```

The call of a function ends when the task is finished or when the timeout has been reached. In the above example, the function ends when the motor is switched on and the control part (regulation) is activated, or when the process takes 10 seconds (in this case, the function returns an error).

For operations where the execution time does not depend on the position controller configuration or on the application, the default timeout can be used by passing the `DSA_DEF_TIMEOUT` (which will be explained later on).

The 'timeout' parameter allows the limitation of the maximum time that the function uses to terminate its task. After this time, if the task has not yet finished, the function ends by returning the `DSA_ETIMEOUT` error.

If the user does not want a timeout, which means there is no time limit to for the synchronous function to execute, the INFINITE value can be assigned to the timeout parameter.

The functions such as `dsa_wait_movement_s()` need a time of execution which depends on the user's application. For functions like these, the use of the default timeout does not make sense.

Remark: As EDI is thread-safe, the user can design a multi-threaded application, each thread using EDI's synchronous functions. This can result in a kind of asynchronous application. However, there is a limit. For each synchronous function, EDI uses an internal synchronous event. The total number of synchronous events is limited to 50.

3.4.2 Asynchronous functions and callbacks

During the development of an application, we mainly use synchronous functions. Nevertheless, in a few cases the use of synchronous functions is not ideal or requires the use of multiple threads. For example, the user wants to monitor the status of a position controller and at the same time wait for the end of a movement. In this case, the use of asynchronous functions can be useful. As mentioned above, the synchronous functions naming end with '_s' and the last parameter is called 'timeout'. There is an equivalent asynchronous function for every synchronous function. Their name ends with an '**_a**' instead of '**_s**' and the last parameter is a pointer to a function and a generic pointer (`void *`) instead of a timeout. For example, here is how to power a motor using the asynchronous version of the implementing function:

```
DSA_CALLBACK do_something_on_poweron_end(DSA_DEVICE_BASE *dev, int err,
                                         void *param);
long[4] params;
...
if (err = dsa_power_on_a(drive,do_something_on_poweron_end,params)) {
   DSA_EXT_DIAG(err, drive);
   goto _error;
}
```

The asynchronous functions start an operation and exit without waiting for the end. Their execution is then really fast. After having started the operation, the user can execute other operations without having to use multiple threads.

When the user calls an asynchronous function, a handler must be provided as parameter which is a pointer to a function often called 'callback'. This function is called by a single thread within the library once the operation is finished.

For example, if the user calls the `dsa_get_register_..._a()` function, it exits before the value of the register is actually returned by the device. Once the value returned, the library calls the callback function, passing the returned register value as parameter.

Besides the handler, the user has the 'param' parameter that can be used at his discretion. This parameter is not interpreted by the library and is passed to the callback function when it is called.

Depending on the EDI function used, different callback types must be used. The different types are:

- `DSA_HANDLER: void callback(DSA_DEVICE_BASE *dev, int err, void *param)`

Must be used on DSA functions which do not return any values like:

```
dsa_power_on_a(DSA_DRIVE_BASE *grp, DSA_HANDLER handler, void *param)
```

- `DSA_INT_HANDLER: void callback(DSA_DEVICE *dev, int err, void *param, int val);`

Must be used on DSA functions which return a single int value like:

```
dsa_get_warning_code_a(DSA_DEVICE *dev, int kind, DSA_INT_HANDLER handler,
                       void *param);
```

- `DSA_LONG_HANDLER: void callback(DSA_DEVICE *dev, int err, void *param, long val);`

Must be used on DSA functions which return a single int value like:

```
dsa_get_register_int32_a(DSA_DEVICE *dev, int typ, unsigned idx, int sidx,
                         int kind, DSA_LONG_HANDLER handler, void *param);
```

- `DSA_EINT64_HANDLER: void callback(DSA_DEVICE *dev, int err, void *param, eint64 val);`

Must be used on DSA functions which return a integer 64 bits value like:

```
dsa_get_register_int64_a(DSA_DEVICE *dev, int typ, unsigned idx, int sidx,
                         int kind, DSA_INT64_HANDLER handler, void
                         *param);
```

- `DSA_DWORD_HANDLER: void callback(DSA_DEVICE *dev, int err, void *param, dword val);`

Must be used on DSA functions which return a dword value like:

```
dsa_get_profile_limit_mode_a(DSA_DRIVE *obj, int kind, DSA_DWORD_HANDLER
                             handler, void *param);
```

- `DSA_FLOAT_HANDLER: void callback(DSA_DEVICE *dev, int err, void *param, float val);`

Must be used on DSA functions which return a single float value like:

```
dsa_get_register_float32_a(DSA_DEVICE *dev, int typ, unsigned idx, int
                           sidx, int kind, DSA_FLOAT_HANDLER handler, void
                           *param);
```

- `DSA_DOUBLE_HANDLER: void callback(DSA_DEVICE *dev, int err, void *param, double val);`

Must be used on DSA functions which return a single double value like:

```
dsa_get_pl_proportional_gain_a(DSA_DRIVE *obj, int kind, DSA_DOUBLE_HANDLER
                               handler, void *param);
```

- `DSA_STATUS_HANDLER: void callback(DSA_DEVICE *dev, int err, void *param, const DSA_STATUS *status);`

Must be used on DSA functions which return a `DSA_STATUS` parameter like:

```
dsa_wait_status_equal_a(DSA_DEVICE *drv, DSA_STATUS *mask, DSA_STATUS *ref,
                        DSA_STATUS_HANDLER handler, void *param);
```

- `DSA_2INT_HANDLER: void callback(DSA_DEVICE *dev, int err, void *param, int val1, int val2);`

Must be used on DSA functions which return 2 integer 32 bits values like:

```
dsa_quick_register_int32_request_a(DSA_DEVICE *dev, int typ1, unsigned
                                   idx1, int sidx1, long *val1, int typ2,
```

```
                                    unsigned idx2, int sidx2, long *val2,
                                    dword *rx_time, DSA_2INT_HANDLER
                                    handler, void *param);
```

- `DSA_PROGRESS: void callback (const char *text, void *puser);`

Must be used on DSA functions which call progress callback like:

```
      dsa_system_configuration_backup (const char *zip_file_name, int nb_url,
                                       const char* urls[], dword flag, dword
                                       optional_block_type_mask, DSA_PROGRESS
                                       progress_fct, void *puser);
```

Remark: EDI callback functions are called by a single internal EDI thread. Therefore, the user callback functions should be as short as possible, avoiding to block the EDI callback mechanism.

Each pending callback uses an internal asynchronous event. The total number of asynchronous events is limited to 50. Therefore, the user should avoid having more than 50 pending asynchronous functions.

3.4.3 Generic functions

All actions that can be done on a device resume to the following three basic operations:
- Sending a command with zero, one or several parameters;
- Setting the value of a register;
- Reading the value of a register.

On top of these three basic operations, there is the status management which will be detailed in §7.

Most of the operations available for the user from the DSA library are special cases of one of these three basic operations. It means that through these three operations it is possible to have access to all the available functionalities in a device. It is for this reason that the DSA library has a set of functions - termed *generic* - for doing these three operations. In fact, there are more than three, not only because there are synchronous and asynchronous versions of each, but also because there are different functions depending on the number and type of the parameters of the commands and depending on whether the value of a given register is wanted in ISO or controller units. Refer to the **"HTML Reference Manual"** for the complete list.

However, the generic functions can be cumbersome to use as they require to know exactly the command's syntax and parameters or the registers that need to be set to implement an operation. Therefore, the EDI package provides functions that implement the specific operation without having to know which command to send or register to read/write. These are called specific functions (refer to next section) and they exist for every operation performed frequently on ETEL devices.

Generic functions remain useful to access the functionalities for which there is not (yet) a specific function available. For example, the DSA library does not have any function to reset the position controller (RSD). To do execute this action, the user has to use a generic function.

Remark: ETEL reserves the right to modify the command number and register number. Specific EDI functions will handle these changes and make them transparent for the user. However, when using generic functions, the user will have to adapt the code to these changes, making it more difficult to maintain. Therefore, ETEL does not recommend the use of generic functions unless strictly necessary.

3.4.4 Specific functions

Operations can be executed on the devices once an associated object has been created and the communication established. Specific functions implement the most often used operations.

Remark: The contents of EDI4.xx was revised and old functions to access controllers of the DSB/DSC families have been removed. Refer to §15.1 for a list of these functions.

For example, to switch on the position controller, the following function must be called:

```
err = dsa_power_on_s(drive1, 10000);
if (err != 0) {
```

```
    printf("problems during power on\n");
    exit(1);
}
```

The first parameter of each function is always the `DSA_DRIVE` or `DSA_MASTER` object which represents the device on which the operation will be executed. The last parameter is always the 'timeout', which represents the maximum time which allowed to complete the operation. This time is always expressed in milliseconds.

The function exits when the operation is finished or when the timeout has elapsed. In the above example, the function exist when the motor is switched on and the control part (regulation) is activated or when the process takes 10 seconds. In this case, the function returns an error.

For operations where the execution time does not depend on the position controller configuration or on the application, the default timeout can be used by giving the `DSA_DEF_TIMEOUT` (which will be explained later on).

Other functions may require more parameters. For example, it is the case for the function which starts a movement, which also requires as input the target point. Here is an example on how a movement can start with a target point at 0.32 meter from the origin point:

```
err = dsa_set_target_position_s(drive1, 0, 0.32, DSA_DEF_TIMEOUT);
if (err != 0) {
   printf("cannot start movement\n");
   exit(1);
}
```

In this example, the movements are specified relative to the software position limits set on the position controller (corresponding to parameters KL34 and KL35 on the AccurET family). So, it is necessary to read these position limits first. This is quite a common operation, so there are specific functions available for this:

```
err = dsa_get_min_soft_position_limit_s(axisX,&pos_min,
                                        DSA_GET_CURRENT,
                                        DSA_DEF_TIMEOUT);
err = dsa_get_max_soft_position_limit_s(axisX, &pos_max,
                                        DSA_GET_CURRENT,
                                        DSA_DEF_TIMEOUT);
```

A list of the available functions is provide in the appendix (see §15.1). If the user is used to program Sequences in the ETEL language or is familiar with the commands on the ETEL terminal, it is easy to find the correspondence between these commands and the functions of the DSA library.

Here are some examples of specific functions:

- `dsa_reset_error_ex_s(drive1, 123, DSA_DEF_TIMEOUT);`
- `dsa_power_on_s(drive1, 10000);`
- `dsa_homing_start_s(drive1, 10000);`
- `dsa_wait_movement_s(drive1, 60000);`
- `dsa_set_target_position(drive1, 0, 0.5, DSA_DEF_TIMEOUT);`

3.5 Error management

In general, each function of the EDI package returns an integer (int) which represents the error code. If the operation ends with success, the function returns 0, otherwise it returns the number of the error represented by a negative number.

The possible error code are defined by a series of #define stored in dsa40.h header file. For the DSA library the possible error codes are as follows:

#define	Error code	Comment
DSA_E_TRAJ_CALCULATION_0	-380	Error Trajectory calculation 0
DSA_E_TRAJ_CALCULATION_1	-381	Error Trajectory calculation 1
DSA_E_TRAJ_CALCULATION_2	-382	Error Trajectory calculation 2
DSA_E_TRAJ_CALCULATION_3	-383	Error Trajectory calculation 3
DSA_E_TRAJ_CALCULATION_4	-384	Error Trajectory calculation 4
DSA_E_TRAJ_CALCULATION_5	-385	Error Trajectory calculation 5
DSA_E_TRAJ_CALCULATION_6	-386	Error Trajectory calculation 6
DSA_E_TRAJ_CALCULATION_7	-387	Error Trajectory calculation 7
DSA_E_TRAJ_CALCULATION_8	-388	Error Trajectory calculation 8
DSA_E_TRAJ_CALCULATION_9	-389	Error Trajectory calculation 9
DSA_EACQDEVINUSE	-337	One of the device is already doing an acquisition
DSA_EACQNOTPOSSIBLE	-336	Drives must be connected with TransnET
DSA_EAXESRECOVER	-309	TransnET established but axes not recovered
DSA_EBADCOMPRESSTOOL	-303	Unhandled compress tool
DSA_EBADDRIVER	-328	Wrong version of the installed device driver
DSA_EBADDRVVER	-325	A drive with a bad version has been detected
DSA_EBADIPOLGRP	-327	The ipol group is not correctly defined
DSA_EBADLIBRARY	-333	Function of external library not found
DSA_EBADPARAM	-322	One of the parameter is not valid
DSA_EBADSEQVERSION	-338	The sequence version is not correct
DSA_EBADSTATE	-324	This operation is not allowed in this state
DSA_EBUSERROR	-313	The underlaying etel-bus is not working fine
DSA_EBUSRESET	-314	The underlaying etel-bus in performing a reset operation
DSA_ECANCEL	-319	The transaction has been canceled
DSA_ECFGCOMPFILE	-339	File has been compiled for a different axes configuration
DSA_ECONVERT	-317	A parameter exceeded the permitted range
DSA_EDRVERROR	-311	Drive in error
DSA_EDRVFAILED	-323	The drive does not operate properly
DSA_EEQUATION	-340	Equation cannot be resolved
DSA_EFLUSH	-341	Flush communication timeout
DSA_EINTERNAL	-316	Some internal error in the etel software
DSA_EMAPNOTACTIVATED	-335	Mapping cannot be activated by the device
DSA_EMETAHEADER	-300	Bad metadata header format
DSA_EMETAHEADERVER	-301	Bad metadata header version
DSA_EMETAPARSER	-304	Metadata parsing error
DSA_ENOACK	-312	No acknowledge from the drive
DSA_ENODRIVE	-320	The specified drive does not respond
DSA_ENOFREESLOT	-330	No free slot available
DSA_ENOFWINPOOL	-305	Firmware not found in FW pool
DSA_ENOLIBRARY	-332	External library not found
DSA_ENOTIMPLEMENTED	-326	The specified operation is not implemented
DSA_EOBSOLETE	-329	Function is obsolete
DSA_EOPENPORT	-321	The specified port cannot be open
DSA_ERTVREADSYNCRO	-331	RTV read synchronization error
DSA_ERTVWRITESYNCRO	-307	RTV write synchronization error
DSA_ESYNTAX	-334	Mapping file syntax error
DSA_ESYSTEM	-315	Some system resource return an error
DSA_ETIMEOUT	-310	A timeout has occurred
DSA_ETRANS	-318	A transaction error has occurred
DSA_ETRANSNET	-308	TransnET is not established in imparted time

#define	Error code	Comment
DSA_EUNZIP	-302	Unable to unzip metadata buffer
DSA_EUPLOADFILE	-306	MD5 of uploaded file does not fit MD5 of original
DSA_EWFP	-342	Communication not opened (device in WaitForProgram)

It is the developer's responsibility to verify these error codes and react in consequence. Good programming practice recommend for each error code to be tested and reacted upon.
The DSA library offers a means of error diagnostic with its `DSA_EXT_DIAG()` macro:

```
int err;
...
if (err = dsa_power_on_s(...)) {
   DSA_EXT_DIAG(err, drv);
   goto _error;
}
```

The `DSA_EXT_DIAG` macro prints on the standard output an error message, the status of the device at the time of the error and the call stack trace (with line numbers) of the part of the execution performed by the EDI package up until the error occurred. This output can be used by the developer to identify the source of a problem. It also gives in depth detailed information on the context and the whereabouts of the source of error which is a very important help to ETEL when supporting a user development. Developers are therefore strongly encouraged to use this call to be able to benefit from an efficient support from ETEL.

Remark: ETEL's support team requires detailed diagnosis information for any issues that are reported.

A detailed description of the meaning of the information outputed by `DSA_EXT_DIAG()` is given in appendix §15.3. The output of `DSA_EXT_DIAG` is the standard output. If the user application does not redirect its standard output or if the standard output is not visible, it is advised to use `DSA_EXT_SDIAG` or even `DSA_EXT_FDIAG` to store the error message.

The user can also test the smooth operation of a DSA function in a more classical way, but less efficient:

```
int err;
...
err = dsa_power_on_s(...);
if (err == 0)
   printf ("power on done.\n");
else if (err == DSA_ETIMEOUT)
   printf("timeout error.\n");
else
   printf("error %d during power on.\n", err);
...
```

The DSA library also offers a function which enables the conversion of an error code into a text error message in the form of a string of characters. Here is the prototype:

```
const char *dsa_translate_error(int code);
```

Here is an example of a way to process the return code using this function:

```
int err;
...
err = dsa_power_on_s(...);
if (err) {
   printf("ERROR %d : %s\n", err, dsa_translate_error(err));
   exit(err);
}
printf("power on done.\n");
...
```

3.6 Memory management

A safe application should be able to run for several days, weeks and even months. One of the well known issues is called memory fragmentation. This problem occurs when dynamic memory is continuously allocated and deallocated. To avoid this problem, it is advised to divide the application in 3 phases:

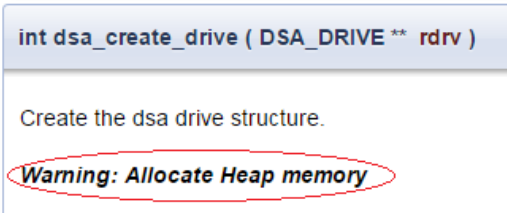
3.6.1 Initialization phase

The initialization phase is done only once at the startup of the application:
- Handle the system configuration (download firmware, sequences, registers…).
- Create the required objects (`dsa_create_drive`, `dsa_create_master`, `dsa_create_acquisition` …).
- Open the communication (`dsa_open_u`…).

=> Dynamic allocation allowed.

3.6.2 Processing phase

The processing phase is generally a loop executing the real job of the application (can run for several months!).
- Check in HTML documentation that none of the used functions allocates memory.



=> Dynamic allocation forbidden.

3.6.3 End phase

The end phase is done once just before the application exits:
- Close the communication (`dsa_close`).
- Destroy the objects (`dsa_destroy`, `dsa_destroy_acquisition`...).

=> Free dynamically allocated resources.

This way, dynamic memory allocation is controlled and no fragmentation occurs. Keep also in mind that dynamic memory allocation can be hidden, especially when using a high level language like C#.

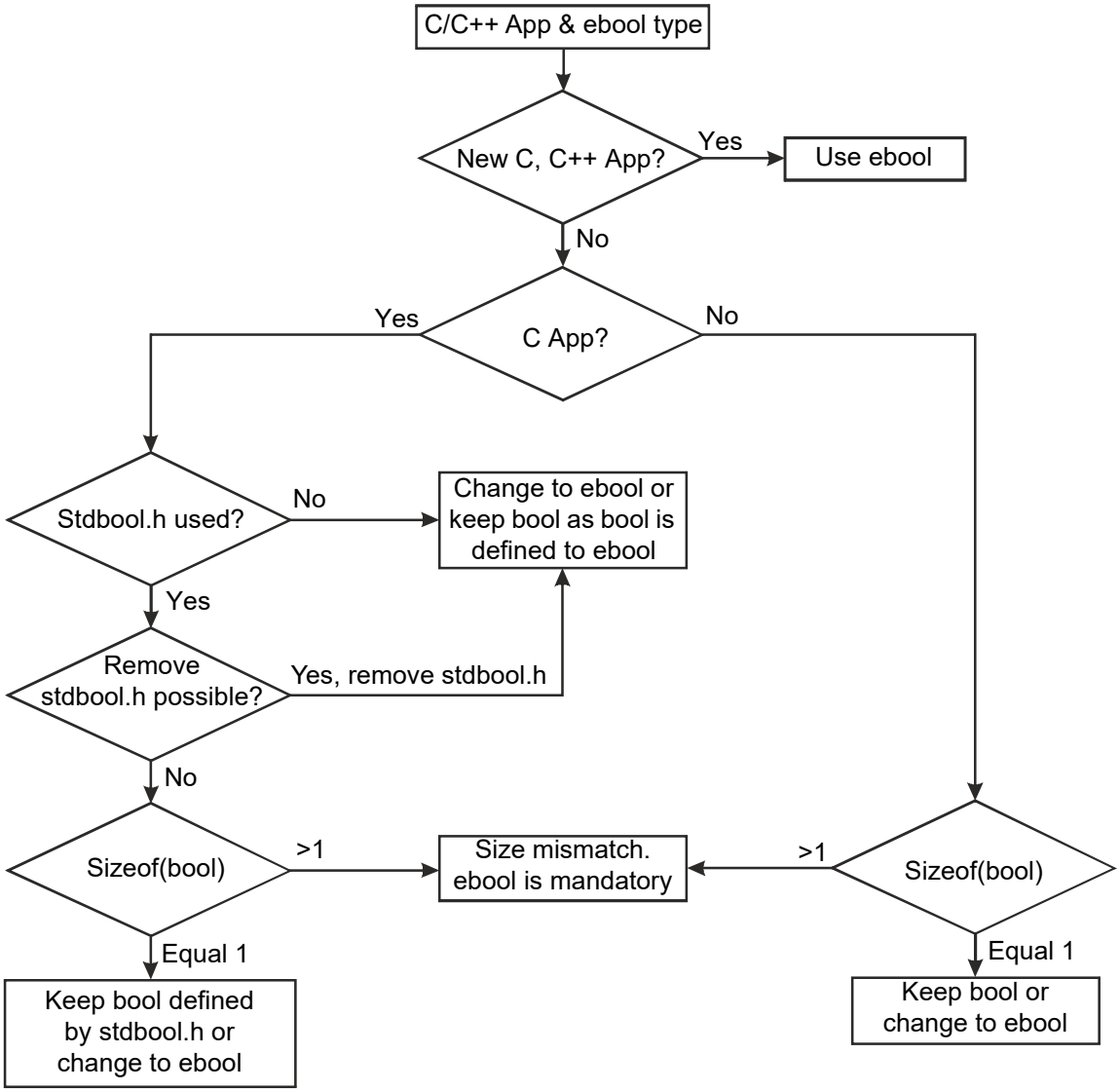
3.7 Boolean type: ebool

Since the release of EDI 4.17A, a new Boolean type is defined named ebool. This type corresponds to an unsigned char with a size of 1 byte.

In the standard for C++ programming language, the boolean type is defined but the sizeof(bool) could be equal or bigger (≥) to 1 byte depending on the C++ compiler that is used. For C language, the boolean type is present from C99 version and is defined through "stdbool.h". As for C++ language, the sizeof(bool) could be equal or bigger (≥) to 1 byte.

Therefore, it is recommended for EDI C/C++ user applications to use the ebool type for any variables of boolean type to avoid any risks linked to size mismatch between C/C++ compilers.

For the C# language, the sizeof(bool) is equal to 1 byte, so there is no risk of size mismatch.



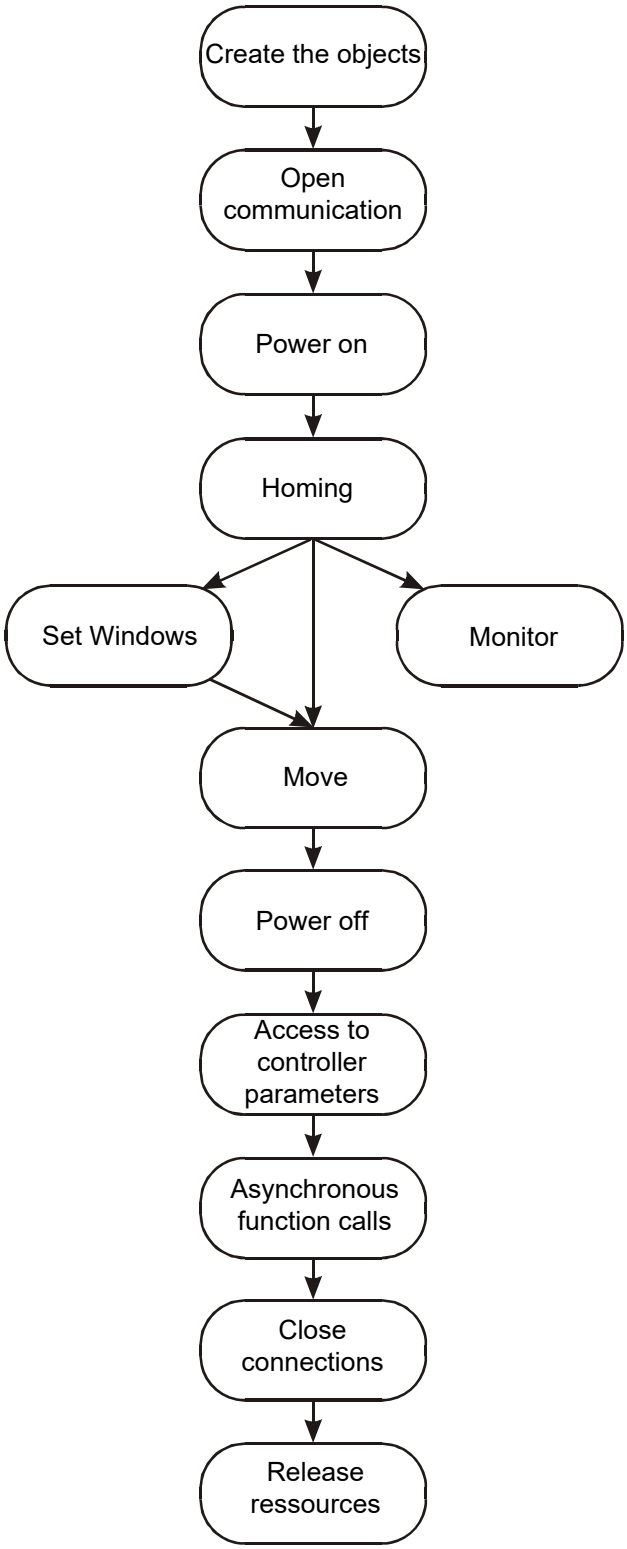
4 Application startup

4.1 Presentation of the general example

The example is given in C as it is a common general basis for programmers and other languages as well.

This example aims at using the library to control two axes concurrently via an UltimET.

EDI Application Example Synopsis



The system commissioning aspects are **outside** the scope of this example, mainly because they are beyond the scope of the DLL itself. Hence the example does **not** cover:

- Downloading firmware onto the UltimET and the position controllers (although there is an example showing how to do it);
- Setting of the position controllers;
- Commissioning can be done through ComET software.

A first set of actions aims at getting the position controllers in an operational state which entails:

- Establishing the communication with the UltimET and the position controllers connected to it (§4.5);
- Powering up the position controllers (§4.6);
- Performing the homing procedure on each position controller (§4.7).

The next set deals with making movements on each axis by first describing how to set precision windows (§5.2), and then actually specifying and starting the movement (§5.3 and §5.4). The movements will be controlled by the UltimET which will execute them in interpolated mode.

In a more elaborate step, one thread will be created to monitor the current position (§6) of the motors. The monitoring thread will show the current position of each motor every 100ms. Another thread will loop indefinitely waiting for a user input: the space bar will immediately stop the movement and set one of the controller's digital outputs.

Additional features will be described in:
- §11 for 'Access to controller parameters';
- §12 for 'Asynchronous function calls';

Finally, it will be shown how to power off the position controllers (§4.6).

4.2 Prerequisites

The first step in using an ETEL controller is to tune its position and other regulation algorithms. This is done prior to programming any functional behavior via ETEL's commissioning tool ComET and is outside the scope of this documentation. It is considered henceforth that these operations have been carried out successfully. They are a necessary prerequisite to start testing a software application developed with the EDI package.

4.3 Application initialization steps

From a purely programming point of view, there are a couple of header files to be included at the beginning of source code of this example:

- some standard libraries:

```
#include <stdio.h>          /* standard I/O library*/
#include <stdlib.h>         /* standard general purpose library*/
#include <ctype.h>          /* standard character conversion library*/
#include <math.h>
```

- the platform specific header for the thread management library:

```
#include <process.h>        /* standard multithreading library*/
```

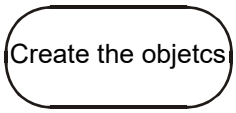
- the EDI top level library header (version 4.x):

```
#include <dsa40.h>
#include <dmd40.h>
```

Also, most functions of the library return an error code or 0 if no error occurred. A variable is needed to store the last error. The error codes are negative values, ranging from -399 to -300 for the dsa40 library:

```
int err = 1;                /* initialized to a value that is NOT */
                            /* returned by the library functions  */
```

4.4 Creating the objects



As an illustration of the creation of objects and for the needs of the example, the objects representing the controllers and the UltimET must be created.

A pointer to a `DSA_DRIVE` object for each position controller must be defined. This hidden object is defined in the dsa40 library. It is not required to access the members of this object directly, but this object has to be passed to various dsa library functions. The pointer must be initialized to NULL before calling the «create drive» function, otherwise this function will fail.

```
DSA_DRIVE *axisX = NULL;
DSA_DRIVE *axisY = NULL;
```

The same goes for the object representing the UltimET:

```
DSA_MASTER *UltimET = NULL;
```

Once the pointers have been declared, the actual objects themselves have to be created and initialized:

```
/* Create the drive and UltimET objects. */
if (err = dsa_create_drive(&axisX)) {
   DSA_EXT_DIAG(err, axisX);
   goto _error;
}
if (err = dsa_create_drive(&axisY)) {
   DSA_EXT_DIAG(err, axisY);
```

```
    goto _error;
  }
  if (err = dsa_create_master(&UltimET)) {
    DSA_EXT_DIAG(err, UltimET);
    goto _error;
  }
```

When the UltimET manages interpolated movements, the commands must be sent to what is called an interpolation group. This interpolation group can be used like a normal device group. All position controllers within this group can also be interpolated. For this, a pointer to a `DSA_IPOL_GROUP` (like a `DSA_DRIVE_GROUP`) object mus be defined. This object can be considered as an array containing `DSA_DRIVE` objects. The client can set, change and retrieve the different devices belonging to this object through some access functions.

Like the `DSA_DRIVE`, thus group pointer must be initialized to NULL before calling the created group function, otherwise this function will fail.

```
DSA_IPOL_GROUP *igrp = NULL;
```

To create the interpolation group object the size for the group must be provided and cannot be changed afterwards.

```
if (err = dsa_create_ipol_group(&igrp, 2)) {
  DSA_EXT_DIAG(err, igrp);
  goto _error;
}
```

In this example, two position controllers are assigned to this group by calling the function `dsa_add_group_item()` that requires two arguments: the group and the position controller to assign.

```
if (err = dsa_add_group_item(igrp, axisX)) {
  DSA_EXT_DIAG(err, axisX);
  goto _error;
}
if (err = dsa_add_group_item(igrp, axisY)) {
  DSA_EXT_DIAG(err, axisY);
  goto _error;
}
```

Associate the UltimET to the interpolation group. When an interpolated movement on a group of axes is required, the dsa40 library will use the UltimET associated with the group.

```
if (err = dsa_set_master(igrp, UltimET)) {
  DSA_EXT_DIAG(err, igrp);
  goto _error;
}
```

The next step consists of opening the communication channels.

4.5 Establishing the communication

Once the device objects have been created, a physical device must be associated with it. The communication bus used to communicate with the position controller, as well as the physical address of the position controller (axis number) must be indicated. This is usually done by the `dsa_open_u` function, which takes the URL of the position controller as the parameter. The URL identifies the protocol used, the communication bus type and characteristics and the corresponding physical axis used. The only protocol recognized by ETEL's EDI package is the proprietary ETB (ETEL Bus) protocol. Hence all URLs used for referencing ETEL devices take the following form:

"**etb**:<bus type>:<axis>"

As a simple example, if we want the position controller `axisX` object to correspond to the axis 12 which is connected to the USB port of the PC, the communication would be established using the following call:

```
err = dsa_open_u(axisX, "etb:usb:12");
```

This function will first open the communication bus on the condition that it is not already opened and then will store the axis number in the `axisX` object. The second parameter is the character string representing the URL.

The general syntax for ETEL URLs is described hereunder:

URL = "**etb** : <communication bus> : <axis number>"

<axis number> is either

an integer, ranging from 0 to 62 for the AccurET family (the axis number is set via the controller's dip switches or using the AXI command; refer to the corresponding "AccurET Operation & Software Manual" for further information);

or «*» to designate the UltimET.

<communication bus> is a string describing the bus used and depends on the type of bus.
The following table describes the syntax of <communication bus>.

Bus type	Syntax	Description
UltimET Light TCP/IP ULTIMET ADVANCED TCP/IP ETND	ETN://<ip-address> :<port>[,<flags>]	Remote access via TCP/IP to : - UltimET Light TCP/IP - ULTIMET ADVANCED TCP/IP - ETND deamon <ip-address> : the IP address of the device or the host running ETND (localhost or 127.0.0.1 can be used) <port>: The IP port to use : - UltimET Light : 1129,1128,1127 - ULTIMET ADVANCED : 1129 (2 connection available) - ETND : depends on the port.properties file present in the directory where ETND is launched. Use the port xxx specified in line port.n.accept.0.driver=ETN:xxx UltimET Light TCP/IP available flags (optional): - T=<keep-alive> - x ULTIMET ADVANCED TCP/IP available flags (optional): - T=<keep-alive> - x ETND available flags (optional): - T=<keep-alive>
UltimET Light PCI/PCIe	ULTIMET [DEVIDX=n][,<flags>]	Direct access to UltimET Light PCI/PCIe DEVIDX=<n> : (optional) - In the case where several UltimET Light are present in the PC, this allow to specify which UltimET must be opened. First UltimET Light is specified with n=0. The order depends of the PCI/PCIe slots numbering. available flags (optional): - r - x
ULTIMET ADVANCED PCI/PCIe	UA [DEVIDX=n][,<flags>]	Direct access to ULTIMET ADVANCED PCIe DEVIDX=<n> : (optional) - In the case where several ULTIMET ADVANCED are present in the PC, this allow to specify which ULTIMET ADVANCED must be opened. First ULTIMET ADVANCED is specified with n=0. The order depends of the PCI/PCIe slots numbering. available flags (optional): - x

Bus type	Syntax	Description
USB	USB	Opening communication on USB will open all ETEL devices connected through USB to the PC. EDI will act as a master. But be aware that the: - devices will not be synchronized as they are when using a motion controller. - each devices must have a unique axis number

Flags explanations:

- T=<keep-alive>
  keep-alive is a number of 500ms packets
  The communication between EDI and Device is kept by sending records each <keep alive> ms. Both EDI & device checks the communication activity and if it detects that after 3 times the <keep alive> time no record is transmitted, it will generate an error and disconnect.
  Setting -1 as keep-alive will disable the check
- x
  Reset all Connected AccurET: All devices present on TransnET are rebooted.
- r
  Reset UltimET: the UltimET processor is resetted and firmware restarted.

Below are some examples of commonly used URLs:

etb:ETN://172.22.7.200:1149:12 => used to open a communication with controller 12 connected to a computer whose IP address is 172.22.7.200 and that is connected to the controllers network via port 1149. The communication between the remote computer and the controller's network for port 1149 will be described in the port.properties of that computer.

etb:ULTIMET,r:* => used to open and reset communication with an UltimET Light PCI/PCIe itself.

etb:ULTIMET:23 => used to open communication with an AccurET controller 23, using the TransnET connected to an UltimET Light in a PCI slot.

etb:USB:2 => used to open communication with an AccurET controller 2, using USB.

etb:ETN://172.22.10.112:1129,T=-1:0 => used to open a communication with controller 0 using TransnET connected to an UltimET TCP/IP whose IP address is 172.22.10.112. The port 1129 is used. T=-1 allows the user to disable keep-alive handshaking between PC and UltimET. This handshaking has been implemented to check the link presence between the PC and the UltimET TCP/IP. Every 10 seconds, a message is sent by EDI to the UltimET. If the UltimET does not receive this message after 3x10 seconds, it falls into error 1603.

etb:UA DEVIDX=1, T=10000 => used to open communication with ULTIMET ADVANCED PCIe. The keep alive is set to 10 seconds, which means that each 10 seconds, a record is sent by EDI to ULTIMET ADVANCED to keep communication alive. If the PC application crashed, ULTIMET ADVANCED will fall in error after 3x10 seconds and will automatically close the connection. If ULTIMET ADVANCED crashed, EDI will close the communication after 30 seconds. Each command sent will then return a `DSA_EBUSSTATE` error.

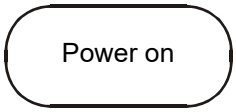
In our example, the following lines explain how to establish the communication. Note that the example is opening communication through an UltimET Light PCI/PCIe. If instead, the communication should be established with an UltimET Light TCP/IP, ULTIMET ADVANCED PCIe or TCP/IP, only the specified URL should be modified, with the rest of the example remaining the same:

```
if (err = dsa_open_u(axisX, "etb:UltimET:0")) {
  DSA_EXT_DIAG(err, axisX);
  goto _error;
}
if (err = dsa_open_u(axisY, "etb:UltimET:1")) {
  DSA_EXT_DIAG(err, axisY);
```

```
    goto _error;
  }
  if (err = dsa_open_u(UltimET, "etb:UltimET:*")) {
    DSA_EXT_DIAG(err, UltimET);
    goto _error;
  }
```

Remark: When opening communication with several devices, it is mandatory to use the same URL for each device (except the axis number). For example, if URL etb:UltimET,rx:* is used to open communication with UltimET Light, the URL etb:UltimET,rx:0 must be used to open communication with AccurET axis 0.

4.6 Powering on (and off)



Different devices take different time to boot up completely. This means that in a system that includes different types of devices (for e.g. position controllers and a motion controller), some devices will be ready while others have not yet finished booting. This can have the effect of putting in error a device that expects another to be ready, but indeed has not yet finished booting. This is why, when all devices have finished booting, it is recommended to start the application by a general «reset error» command:

```
if (err = dsa_reset_error_ex_s(igrp, 123, DSA_DEF_TIMEOUT)) {
  DSA_EXT_DIAG(err, igrp);
  goto _error;
}
```

Now, commands can be sent to the controllers. The first thing to do is to put the position controller in power on state. This is done via the `dsa_power_on_s` command. It is the equivalent to the controller's `PWR=1` command.

The affix "_s" on the function's name indicates that this is a «synchronous» function. Synchronous functions wait until the end of the operation before returning. All synchronous functions have a timeout parameter as the last parameter. This parameter orders the function to return with a timeout error (`DSA_ETIMEOUT`) if no response comes from the position controller before the end of the specified timeout. This lack of response usually indicates an error in the application, or could result from bad position controller parameters. An appropriate timeout value depends on the application and the command issued. For e.g. in the «power on» case, less than 1 second could be appropriate if the phasing initialization is based on current pulses (K90 = 1), but more than 5 seconds could be required if the phasing initialization is based on constant current (K90 = 2).

This can be done in one of two ways:

- Either switch the power on in each motor individually:

```
if (err = dsa_power_on_s(axisX, 10000)) {
  DSA_EXT_DIAG(err, axisX);
  goto _error;
}
if (err = dsa_power_on_s(axisY, 10000)) {
  DSA_EXT_DIAG(err, axisY);
  goto _error;
}
```

- Or, since there is a group object (primarily for interpolation purposes, but it can also be used anywhere a group is expected), send the command to the group in general, the DLL taking upon itself the responsibility of dispatching it to all the members of the group:

```
if (err = dsa_power_on_s(igrp, 10000)) {
  DSA_EXT_DIAG(err, igrp);
  goto _error;
}
```

Powering off is just as simple as:

```
if (err = dsa_power_off_s(igrp, 10000)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

4.7 Homing

The next step in the preparation of the controllers and motors for operation consists on performing the homing procedure in order to find the reference for the motor's absolute position. Once again, this can be done on each axis individually:

```
if (err = dsa_homing_start_s(axisX, 10000)) {
   DSA_EXT_DIAG(err, axisX);
   goto _error;
}
if (err = dsa_homing_start_s(axisY, 10000)) {
   DSA_EXT_DIAG(err, axisY);
   goto _error;
}
```

or on the group as a whole:

```
if (err = dsa_homing_start_s(igrp, 10000)) {
   DSA_EXT_DIAG(err, axisY);
   goto _error;
}
```

`dsa_homing_start_s()` is the equivalent of the controller's `IND` command.

This function «only» starts the homing procedure. Before other movements can be executed, the homing procedure must have terminated. So, to wait until the homing is concluded, proceed as follows:

```
if (err = dsa_wait_movement_s(igrp, 60000)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

Remark: Be sure to use a `DSA_GANTRY` object when working with a Gantry in 'Level 1'. In this special case, EDI will forward the homing procedure to the UltimET.

5 Movements

5.1 Introduction to the example's trajectory

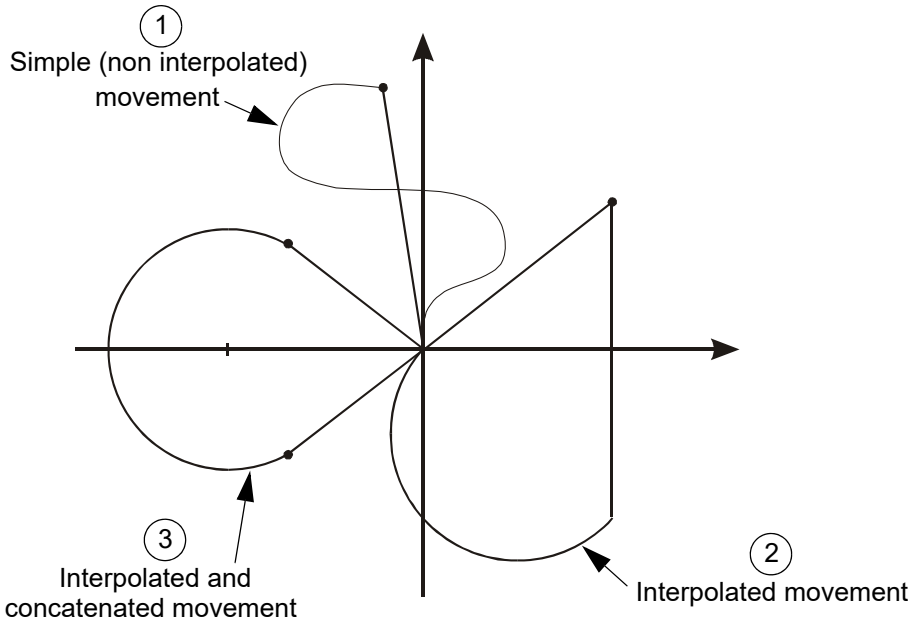
In our example, to make sure the movement never exceeds the limits and that a proper error is generated on the position controller in case this occurs, the coordinates of the positions are always given with reference to the minimum and maximum position limits. These are stored on the controller's parameters KL34 and KL35 for the AccurET-family and can be read as follows:

```
#define AXIS_NB_X 0
#define AXIS_NB_Y 1

double pos_min[2], pos_max[2];

if (err = dsa_get_min_soft_position_limit_s(axisX, &pos_min[AXIS_NB_X],
                                            DSA_GET_CURRENT, DSA_DEF_TIMEOUT)) {
  DSA_EXT_DIAG(err, axisX);
  goto _error;
}
if (err = dsa_get_max_soft_position_limit_s(axisX, &pos_max[AXIS_NB_X],
                                            DSA_GET_CURRENT, DSA_DEF_TIMEOUT)) {
  DSA_EXT_DIAG(err, axisX);
  goto _error;
}
```

The same is repeated for the other axis, storing the available range of motion in a variable to be used later:

```
double range_of_motion[2];
range_of_motion[AXIS_NB_X] = pos_max[AXIS_NB_X] - pos_min[AXIS_NB_X];
range_of_motion[AXIS_NB_Y] = pos_max[AXIS_NB_Y] - pos_min[AXIS_NB_Y];
```

In our example the motors will move so that the toolpoint executes the trajectory depicted next:



5.2 Defining the position windows

Set Windows

To ensure that the movement complies with the application's requirements, there are several windows that have to be (or can be) defined, depending on what type of requirement is governing the application. For example, for a position driven application, the user might want to constrain the tracking error, and/or the position settling time, and/or the precision of the target position.

The EDI package provides several specific functions for this purpose:

- `dsa_set_following_error_window_s` sets the tracking error limit (position controller parameter K30)
- `dsa_set_position_window_time_s` defines the minimum time during which the position must be within given bounds of the target position for it to be considered reached (position controller parameter K38).
- `dsa_set_position_window_s` sets the acceptable error on the real position (position controller parameter K39) compared to the target position.

Usually, these parameters are set at the position controller's level when integrating the system, but some applications may need to change, for example, the tracking error detection limit depending on an operating mode. This could be done as follows:

```
long axisX_trackingErrorLimit =...;
long axisY_trackingErrorLimit =...;
if (err = dsa_set_following_error_window_s(axisX, axisX_trackingErrorLimit,
                                           DSA_DEF_TIMEOUT)) {
  DSA_EXT_DIAG(err, axisX);
  goto _error
}
if err = dsa_set_following_error_window_s(axisY, axisY_trackingErrorLimit,
                                          DSA_DEF_TIMEOUT)) {
  DSA_EXT_DIAG(err, axisY);
  goto _error
}
```

5.3 Simple (non interpolated) movements

Move

5.3.1 Defining the movement profile

As described in the "AccurET's Operation & Software Manual", the movement can be controlled in a variety of ways depending on its operating mode: force reference, speed reference, external reference or position reference. The controller's mode (position controller parameter K61) is initialized at startup, but can be modified using `dsa_set_drive_control_mode_a()`. Refer to the "**HTML Reference Manual**" for the details of this function.

When in position controlled mode, which is the default mode, the movement can be defined by means of a set of specific functions. In the other operating modes, the movement must be defined using the generic functions (refer to §10.2 for more information).

A position controlled movement can be defined by up to four parameters. One usually sets at least the target position (refer to §5.3.2); the speed, acceleration and jerk time describing the movement profile can also be modified.

To set the movement speed:

```
double profileSpeed = 0.1; /* m/s or t/s for rotary motors */
if (err = dsa_set_profile_velocity_s(axisX,0, profileSpeed, DSA_DEF_TIMEOUT)) {
  DSA_EXT_DIAG(err, axisX);
  goto _error;
}
```

To set the movement acceleration:

```
double profileAcc = 10.0; /* m/s^2 or t/s^2 for rotary motors*/
if (err = dsa_set_profile_acceleration_s(axisX, 0, profileAcc,
                                         DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, axisX);
   goto _error;
}
```

To set the movement jerk time:

```
double jerkTime = 0.01; /* seconds */
if (err = dsa_set_jerk_time_s(axisX, 0, jerkTime, DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, axisX);
   goto _error;
}
```

5.3.2 Starting

Making a single position controller simply go to a given position is quite straightforward. To move to a random position within the bounds of the motor stroke:

```
double position =
pos_min[AXIS_NB_X]+(rand()*(range_of_motion[AXIS_NB_X]) / RAND_MAX;
if (err = dsa_set_target_position_s(axisX, 0, position, DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, axisX);
   goto _error;
}
```

and then wait until the movement is finished:

```
if (err = dsa_wait_movement_s(axisX, 60000)) {
   DSA_EXT_DIAG(err, axisX);
   goto _error;
}
```

The `dsa_set_target_position_s`() is the equivalent to the controller's `POS` command. It sets the value of the target position and starts the movement.

A special note must be made for the second parameter («0» in the above example). The function takes a subindex parameter as its second argument. If a zero is written in the subindex, like above, the movement will start immediately. A value of 1 to 3 in the subindex just prepares the movement, which will then start with the "new set point" command:

```
if (err = dsa_new_setpoint_s(axisX, preparedsIndex, flags ,DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, axisX);
   goto _error;
}
```

Note that this function takes a group as first parameter. Indeed preparing movements at different subindices and using the `dsa_new_setpoint_s` function is meaningful when several position controllers need to start simultaneously. With TransnET, as in this example, all position controllers are synchronized together. In this case, the movements start exactly at the same time (within one microsecond).

The third param is a mask and should contain the logical sum of all the following constants, specifying which information must be fetched from the buffer. The constants include (refer to the **"HTML Reference Manual"** for complete range of values):

- `DSA_STA_POS`: use the target position of the given buffer;
- `DSA_STA_SPD`: use the profile velocity of the given buffer;
- `DSA_STA_ACC`: use the profile acceleration of the given buffer;
- `DSA_STA_JRK`: use the profile jerk of the given buffer.

For example, if subindex 2 was used to store the new target position and movement profile speed, then, to start the movement we would write:

```
if (err = dsa_new_setpoint_s(igrp, 2, DSA_STA_POS | DSA_STA_SPD,
                             DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

5.4 Interpolated movements

In this example, there are two axes and an UltimET, so it is possible to perform interpolated movements.

Interpolated movements are defined by high level commands that are sent to the UltimET which interprets them to generate the set points for each individual position controller making up the interpolation group in order to execute the specified trajectory.

To start making interpolated movements, it is required to to enter into the «interpolated» mode using the `dsa_ipol_begin_s()` function. Once in this mode, one must use the interpolation functions on the position controllers belonging to the interpolation group. They begin with `dsa_ipol_...()`. It is no longer possible to set a target position using `dsa_set_target_position_s()` on a given axis. To do so, it is required to to leave the interpolation mode using the `dsa_ipol_end_s()` function.

Interpolated mode functions allow to specify geometric definitions of the trajectory, such as executing a line, a circle and so on. Refer to the "**HTML Reference Manual**" for the full list of available functions. The current position when entering the interpolation mode is used as reference point for all subsequent movement definitions. When leaving the interpolation mode, the reference reverts to what was defined before.

So, to enter interpolation mode, use:

```
if (err = dsa_ipol_begin_s(igrp, DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

Remark: Initiating the interpolation mode changes the reference point for the position. Whereas before, all positions were relative to the encoder reference mark, once in interpolation mode, the reference point used is the position where the system was when the `dsa_ipol_begin_s()` function was executed. So all coordinates given in interpolated mode are relative to the position at the start of the interpolation mode.

5.4.1 Defining the movement profile

Then, in this mode, it is the maximum tangential speed (in m/s) and accelerations (in m/s^2) that must be defined.

To set the tangential speed of the trajectory:

```
if (err = dsa_ipol_tan_velocity_s(igrp, 0.05, DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

To set the tangential acceleration of the trajectory:

```
if (err = dsa_ipol_tan_acceleration_s(igrp, 0.1, DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

To set the tangential deceleration of the trajectory:

```
if (err = dsa_ipol_tan_deceleration_s(igrp, 0.1, DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

Now we are ready to make our system move.

5.4.2 Executing movements

At this point one can, for example, start the trajectory with a linear segment. The following function defines and starts the movement from the previous end of segment and stops the system at the end of the specified segment.

To specify the coordinates of the end point of the segment:

```
double endPointX = range_of_motion[AXIS_NB_X]/4.0;
double endPointY = range_of_motion[AXIS_NB_Y]/4.0;
```

And to actually go from the end position of the previous segment to the specified end position:

```
if (err = dsa_ipol_line_2d_s(igrp, endPointX, endPointY, DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

After this other lines can be specified by other calls to the same function:

```
endPointX = range_of_motion[AXIS_NB_X] / 4.0;
endPointY = -range_of_motion[AXIS_NB_Y] / 4.0;
if (err = dsa_ipol_line_2d_s(igrp,endPointX, endPointY, DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

or draw a portion of a circle with calls like:

to define the center point of the arc:

```
double centerX = range_of_motion[AXIS_NB_X]/8.0;  /* X of center of arc */
double centerY = -range_of_motion[AXIS_NB_X]/8.0; /* Y of center of arc */
```

to define end point of the arc:

```
double arcendX = 0.0; /* X coordinate of end point of arc (return to center)*/
double arcendY = 0.0; /* Y coordinate of end point of arc (return to center)*/
```

and to execute the movement:

```
if (err = dsa_ipol_circle_cw_c2d_s(igrp, arcendX, arcendY, centerX, centerY,
                                   DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

which will draw an arc ending at the position given by the 2nd and 3rd parameters, centered on the point with the coordinates given by the 4th and 5th parameters (remember, all with respect to the point at which the system was when entering the interpolation mode) and moving in the clockwise direction.

5.5 Interpolated and concatenated movements

Up to now the system would have stopped at the end of each segment. To make a continuous movement, it is necessary to start the concatenation of the segments as follows:

```
if (err = dsa_ipol_begin_concatenation_s(igrp, DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

But, this function should only be executed **after** the first segment has started, to inform the trajectory generator to look ahead for the next segment.

With the concatenation enabled, all segments are processed at constant speed, without stopping between segments. This means that when the interpolator processes one segment, it does not decrease the speed at the end of the segment. It jumps to the next segment without a speed change. The following calls will generate a movement along a trajectory at a constant speed:

```
double endPointX = -range_of_motion[AXIS_NB_X]/8.0;
double endPointY = range_of_motion[AXIS_NB_Y]/8.0;

if (err = dsa_ipol_line_2d_s(igrp,
                             endPointX,    /* Segment end X */
                             endPointY,    /* Segment end Y */
                             DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
if (err = dsa_ipol_begin_concatenation_s(igrp, DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
if (err = dsa_ipol_circle_ccw_c2d_s(
                           igrp,
                           endPointX,                        /* Arc end X */
                           -endPointY,                       /* Arc end Y */
                           -range_of_motion[AXIS_NB_X]/4.0, /* Arc center X */
                           0.0,                              /* Arc center Y */
                           DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
if (err = dsa_ipol_line_2d_s(igrp,
                             0.0,
                             0.0, /* return to interpolation mode origin */
                             DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

Of course, in this mode, one must take care of not introducing angles in the trajectory to avoid damaging the mechanical system.

Once the concatenated segments section is terminated, it is possible to revert to individual segments, in between which the UltimET will drive the system to a stop and a restart:

```
if (err = dsa_ipol_end_concatenation_s(igrp, DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

When the system no longer needs to perform interpolated movements, the interpolation mode is exited by calling:

```
if (err = dsa_ipol_wait_movement_s(igrp, 60000)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

to wait for all segment to be finished, and then:

```
if (err = dsa_ipol_end_s(igrp, DSA_DEF_TIMEOUT)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

to leave the interpolation mode and come back to the previous reference system.

6 Monitoring data

Move

In this section, it will be shown how to read data in a cycle from the controllers. The example will monitor the motor's position and the position controller's status, which are two of the quite common monitoring requirements.

To get feedback data on a regular basis, it is usual to create a specific thread to handle the data. Creating a thread is system and language dependent. On Windows and in C, a thread is created as follows:

```
if (_beginthread(display_thread, 0, igrp) <= 0) {
  err = DSA_ESYSTEM;
  printf("ERROR in file %s, at line %s %s\n",_FILE_,_LINE_,
        dsa_translate_edi_error(err));
  goto _error;
}
```

Notice here a slightly different way to handle the error code. This is because `_beginthread` is a C library call that does not return known EDI error codes. So if there is a system error creating the thread, it is translated into an EDI system error and processed for display.

`_beginthread()` is an operation made available by `<process.h>`. The tasks the thread will do are implemented by the function passed as first parameter, i.e. `display_thread()`. This function will be called by the thread with the third parameter of `_beginthread` as its own parameter; here a pointer to our group of position controllers.

Let's now see what `display_thread()` does.

First, the access to the position controllers must be renewed, which is done with the help of the drive group passed as parameter to the function:

```
DSA_DRIVE_GROUP *grp = (DSA_DRIVE_GROUP *)param; /* get group from paramter*/
DSA_DRIVE *drv[2]; /* create local drive objects*/
for (i = 0; i < 2; i++) {
  if (err = dsa_get_group_item(grp, i, &drv[i])) {
    DSA_EXT_DIAG(err, drv[i]);
    goto _error;
  }
}
```

The task is an infinite loop that will be stopped when the system terminates the whole process after it has come to its end. Every 100ms, the loop will read the position controller status and the actual position for display. Before the position controller status can be read, as it is a DSA object, it must be initialized using the appropriate function. The position controller status is kept up-to-date in the process memory by the DSA library because it is very frequently needed. Therefore, reading the controller's status is a very efficient operation that does not degrade the overall performance of the user's application.

```
for(;;) {
  double pos[2];

DSA_STATUS status[2]= {{sizeof(DSA_STATUS)},{sizeof(DSA_STATUS)}};
for (i = 0; i < 2; i++) {
  int err;
  if (err = dsa_get_status(drv[i], &status[i])) {
    DSA_EXT_DIAG(err, drv[i]);
    goto _error;
  }
}
for (i = 0; i < 2; i++) {
```

```
      int err;
      if (err = dsa_get_position_actual_value_ex_s(drv[i], 0, &pos[i],
                                                   DSA_GET_CURRENT,
                                                   DSA_DEF_TIMEOUT)){
        DSA_EXT_DIAG(err, drv[i]);
        goto _error;
      }
    }
    /*
    * We can now print the status string on the display.
    */
    printf("%04d: ", ++counter);
    for (i = 0; i < 2; i++) {
      printf("AXIS %d: %c%c%c/%4.4fmm", i,
             status[i].drive.moving ? 'M' : '-',
             status[i].drive.warning ? 'W' : '-',
             status[i].drive.error ? 'E' : '-',
             pos[i] * 1.0E3);
      printf((i == 0) ? ", " : "\r");
    }

    /* wait 100ms: system dependent call */
    { extern_stdcall Sleep(int); Sleep(100); }

  }/*end endless loop*/
```

7 Status handling

7.1 Principle

Each ETEL's device has a set of information which allows the user to know which is the state of the device. This information is represented by a bit field. Each bit corresponds to a state (power on, error, warning, movement,...). The value of the bit shows if the device is or is not in this state.

Some of the states found on most ETEL devices, include:

Present	Shows if the device is present (active)
Warning	Shows if a warning appeared on the device
Error	Shows if the device is in error
Moving	Shows if the movement is in progress
In_window	Shows if the motor is in position, in the given window
Sequence	Shows if the sequence is running
Trace	Shows if the acquisition of a trace is in progress
User 0	At the user's disposal
...	
User 16	

The main special feature of the status is its automatic update in real time. Indeed, because the status information needs to be queried often in applications, the DSA library maintains an up-to-date real-time value of the status of each position controller connected to the bus. Therefore, the status access functions do not generate any communication over the bus and are, thus, not time consuming (i.e. these functions can be called frequently without degrading the overall performance of the application).

The user has functions allowing to wait for a device to be in a given state. Thus, the user can be informed of a status change without polling the status.

Here is the list of the available functions:

dsa_get_status	Returns the status of the device
dsa_grp_get_and_status	Applies a bitwise AND on status of all devices of device group and returns it
dsa_grp_get_or_status	Applies a bitwise OR on status of all devices of device group and returns it
dsa_wait_status_equal_s() / _a()	Waits for a device to be in a given state
dsa_wait_status_not_equal_s() / _a()	Waits for a device to quit a given state
dsa_grp_wait_and_status_equal_s() / _a()	Waits for all the devices of a group to be in a given state
dsa_grp_wait_and_status_not_equal_s() / _a()	Waits for all the devices of a group to leave a given state
dsa_grp_wait_or_status_equal_s() / _a()	Waits for one of the devices of a group to be in a given state
dsa_grp_wait_or_status_not_equal_s() / _a()	Waits for one of devices of a group to leave a given state
dsa_cancel_status_wait	Unblocks all status waiting for specified device. The callbacks of asynchronous wait will be called
dsa_grp_cancel_status_wait	Unblocks all status waiting for specified device group. The callbacks of asynchronous wait will be called

7.2 Working with DSA_STATUS

The DSA_STATUS is a union of the following members:

```
typedef union DSA_STATUS {
  int size;                          /**< The size of this structure */
  DsaStatusSWMode sw;                /**< Status for SW1/SW2 access. Use only
                                     when status is got from
                                     Dsa_get_status_drom_drive. Allow access to
                                     homing and init bit field */
  DsaStatusRawMode raw;              /**< Status for raw access. Use carefully.
                                     Compatibility not granted along EDI
                                     versions */
  DsaStatusDriveBitMode drive;       /**< Status for drive bit access */
  DsaStatusDsmaxBitMode dsmax;       /**< Status for dsmax bit access */
  DsaStatusUltimETBitMode ultimet;   /**< Status for UltimET bit access */
  DsaStatusUltimETBitMode master;    /**< Status for UltimET bit access */
} DSA_STATUS;
```

Depending on how the status is read, the user has to access the structure using the corresponding sub-structure. There are several EDI functions which can be used to read the status:

- **The functions accessing the internal realtime status of EDI:**

```
dsa_get_status(DSA_DEVICE *dev, DSA_STATUS *status)
dsa_grp_get_and_status(DSA_DEVICE_GROUP *grp, DSA_STATUS *status)
dsa_grp_get_or_status(DSA_DEVICE_GROUP *grp, DSA_STATUS *status)
dsa_gantry_get_and_status(DSA_GANTRY *gantry, DSA_STATUS *status)
dsa_gantry_get_or_status(DSA_GANTRY *gantry, DSA_STATUS *status)
```

These functions (without timeout parameter) are accessing the internal status of EDI. For reminder, EDI gets and stores in real-time the status of each connected device using M63. Executing these functions **does not generate any communication** between EDI and the device and this is why they do not require any timeout parameter.

When using these functions, the user has to access the read status using one of the following DSA_STATUS member:

```
typedef union DSA_STATUS {
  int size;                          /**< The size of this structure */
  DsaStatusSWMode sw;                /**< Status for SW1/SW2 access. Use only
                                     when status is got from
                                     dsa_get_status_drom_drive. Allow access to
                                     homing and init bit field */
  DsaStatusRawMode raw;              /**< Status for raw access. Use carefully.
                                     Compatibility not granted along EDI
                                     versions */
  DsaStatusDriveBitMode drive;       /**< Status for drive bit access */
  DsaStatusDsmaxBitMode dsmax;       /**< Status for dsmax bit access */
  DsaStatusUltimETBitMode ultimet;   /**< Status for UltimET bit access */
  DsaStatusUltimETBitMode master;    /**< Status for UltimET bit access */
} DSA_STATUS;
```

Use the DsaStatusDriveBitMode drive member if the specified device is an AccurET.
Use the DsaStatusUltimETBitMode ultimet (or master) member if the specified device is an UltimET.
The DsaStatusDsmaxBitMode dsmax has been kept for compatibility reasons but does not have all usable status bits defined.

- **The function accessing the detailed status of the drive:**

```
dsa_get_status_from_drive(DSA_DEVICE *dev, DSA_STATUS *status, long timeout)
```

This function (with a timeout parameter) is reading the status of the AccurET. They are reading the M60 and M61 registers. Therefore, executing these functions **generates communication** between EDI and the device and this is why they have a timeout parameter.

When using this function, the user has to access the read status using the following `DSA_STATUS` member:

```
typedef union DSA_STATUS {
  int size;                          /**< The size of this structure */
  DsaStatusSWMode sw;                /**< Status for SW1/SW2 access. Use only
                                     when status is got from
                                     dsa_get_status_drom_drive. Allow access to
                                     homing and init bit field */
  DsaStatusRawMode raw;              /**< Status for raw access. Use carefully.
                                     Compatibility not granted along EDI
                                     versions */
  DsaStatusDriveBitMode drive;       /**< Status for drive bit access */
  DsaStatusDsmaxBitMode dsmax;       /**< Status for dsmax bit access */
  DsaStatusUltimETBitMode ultimet;   /**< Status for UltimET bit access */
  DsaStatusUltimETBitMode master;    /**< Status for UltimET bit access */
} DSA_STATUS;
```

It is interesting to use this function on an AccurET (not on an UltimET). If you look at the sub-structure definitions, you can see that DsaStatusSWMode member has 2 interesting bits defined:

`init_done`: The initialization procedure has been done
`homing_done`: The homing procedure has been done

These 2 bits are not present in M63. As the EDI real-time status mechanism is done using M63, these bits are not defined in the other sub-structure. **Therefore, you should NOT access the `DSA_STATUS` using the DsaStatusSWMode member when the status is retrieved using `dsa_get_status` function, because some bits are not up-to-date.**

There are 3 ways of initializing the `DSA_STATUS` structure:

- `DSA_STATUS st = {sizeof(DSA_STATUS)};`
  Be carefull, on some OS (like QNX6), this way of initialization does not set the other structure members to 0 (on these kinds of OS, the second method is preferred, especially when it is requested to have the other members set to 0).

- `DSA_STATUS st;`
  `DSA_INIT_STATUS(st);` From EDI-4.11A ONLY

- `DSA_STATUS st;`
  `memset(&st, 0, sizeof(st)); st.size = sizeof(st);`

Here is a reading example of the status:

```
int err;
DSA_STATUS status= {sizeof(DSA_STATUS)};/* very important ! */

...

err = dsa_get_status(drive1, &status);
if (err != 0) {
  printf("I cannot get the drive status : %s\n",
  dsa_translate_error(err));
}

if (status.drive.moving)
  printf("The drive is moving\n");
else
  printf("The drive is stoped\n");
```

In this example, the user has to set the size field of the structure. Given that the status of a position controller si being queried, the status bits can be accessed via the 'drive' structure.

Here is an example where the application waits for the end of a movement by using the waiting on the status:

```
DSA_STATUS mask = {sizeof(DSA_STATUS)};
DSA_STATUS ref = {sizeof(DSA_STATUS)};

...

/* start a movement */
dsa_set_target_position_s(drive1, 0, 1.0, DSA_DEF_TIMEOUT);

/* wait for the end of the movement
* this code is equivalent to dsa_wait_movement_s(drive1, 20000) */
mask.drive.moving = 1;
dsa_wait_status_equal_s(drive1, &mask, &ref, NULL, 20000);
```

In this example, the user waits for the movement to be finished (i.e. 'moving' bit is equal to 0). It must be specified that the user waits for the 'moving' bit to be equal to 0. This is why there are a mask ('mask' variable) and a reference ('ref' variable). In the mask, the bits on which the waiting is done are set to 1. In the example, it is the 'moving' bit. The values the user is waiting for are set in the reference. In the example, the value is equal to 0.

In the mask, all bits except the 'moving' bit must be equal to 0. It is done thanks to the `sizeof(DSA_STATUS)`, which initializes to the first member of the structure and the other members to zero. The 'ref' variable is also set to 0 by this function even if it is not necessary. Only the value of the 'moving' bit is interesting in this variable. The size fields of both structures must be correctly initialized.

In the following example, the user monitors the position controller to determine if it is in error mode or not. To do so, an asynchronous wait is used on the error bit of the status:

```
void DSA_CALLBACK err_handler(DSA_DEVICE *dev, int err,
                              void *param, const DSA_STATUS *status)
{
  printf("ERROR ON THE DRIVE !\n");
}

int main()
{
  DSA_STATUS mask = {sizeof(DSA_STATUS)};
  DSA_STATUS ref = {sizeof(DSA_STATUS)};

  DSA_DRIVE *drive1 = NULL;
  /* create and open the drive */
  dsa_create(&drive1);
  dsa_open_u(drive1,...);

  /* wait for an error - the program do not block here */
  mask.drive.error = 1;
  ref.drive.error = 1;
  dsa_wait_status_equal_a(drive1, mask, ref, err_handler, NULL);

  /* execute movements */
  dsa_set_target_position_s(drive1,...);
  ...

  /* close and destroy */
  dsa_close(drive1);
  dsa_destroy(&drive1);
}
```

7.3 Performances

The status are the fastest way for a device to give a Boolean type information to the PC. This information can be a change of state, an event taking place or any other information. The user can use part of the status to for its own purposes.

The routing of these status must be carefully done. The processing at the PC level is realized by interrupt. A task of the PC can be awakened by a status change. Thus, all the waits on the status do not use the time of the CPU.

Each device has a large number of Boolean information which are often stored in the M60, M61 and M63 monitoring registers. The status do not provide all the information, but only those that require a fast routing to the PC. For example, there is a bit (number 2) of M60 which indicates if the position controller has done the homing or not. This information is obviously useful, but it does not require to be know in real time, therefore, it is not include in EDI's status. If the user wants to know this information, it can be simply request to the position controller the value of M60 using the `dsa_get_status_from_drive_s` function, and access this bit using DsaStatusSWMode member of `DSA_STATUS`.

The status are updated every TransnET cycle time. The motion controller receives an update of the status at this rate (but because some position controllers have a slower status update cycle time, the motion controller will receive during several TransnET cycles the same status value for these position controllers). Each time it detects a change of status, the motion controller informs the PC. Thus, the PC is informed of a status change at the same rate as the motion controller, but is often limited by the performances of the Windows OS.

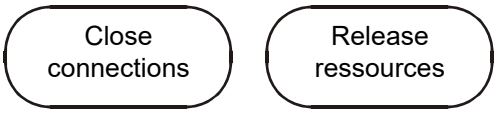
If the performances of the PC do not allow it to monitor the evolution of the status, certain changes of these status can pass undetected at the PC level. For example, if a very short movement is done, the 'moving' bit of the status will be equal to 1 during a very short time. This change can pass unnoticed by the PC and this bit would be considered equal to 1 at all times.

On the other hand, the PC will be always informed of the last state available. If the user wants to wait for the end of the movement, he has to wait for the 'moving' bit to be equal to 0. It is true whatever the performances of the PC. The waiting functions on the status do not wait for a change of state (edge detection) but simply wait for a given state (level detection).

7.4 User status

Among the bits of the status, 16 of them are at the user's disposal. At the PC level, these bits are managed exactly like the others. Thus, the user can read them and wait for them to be in a particular state thanks to the functions described above. At the device level, several other features, such as error, in window, interpolation marks, K177 user status parameter modify the state of these bits. See the "User's Manual" of the related device to know how these interact.

8 Terminating the application



Application termination operations take place in the reverse order of those described in §4 which correspond to creating the objects and establishing the communication.

These two functions allocate resources. During the creation of an object or a group, memory is allocated. When establishing the communication, for e.g. the use of the USB port, the interrupt IRQ or other resources specific to the communication bus are assigned as well. These resources must be released as soon as there are not used any more. It is recommended to do it just before exiting the application.

The `dsa_close()` function closes the communication and the `dsa_destroy()` function destroys an object or a group. Closing the communication must be done before the destroying the object.

Firstly, the connections must be closed:

- connection to the UltimET:

```
if (err = dsa_close(UltimET)) {
   DSA_EXT_DIAG(err, UltimET);
   goto _error;
}
```

- connections to both position controllers:

```
if (err = dsa_close(axisX)) {
   DSA_EXT_DIAG(err, drv);
   goto _error;
}
if (err = dsa_close(axisY)) {
   DSA_EXT_DIAG(err, drv);
   goto _error;
}
```

Then, just like the objects were created at the beginning of the application, the memory allocated to them must be released:

- group object:

```
if (err = dsa_destroy(&igrp)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
```

- UltimET object:

```
if (err = dsa_destroy(&UltimET)) {
   DSA_EXT_DIAG(err, UltimET);
   goto _error;
}
```

- and the drive objects

```
if (err = dsa_destroy(&axisX)) {
   DSA_EXT_DIAG(err, axisX);
   goto _error;
}
if (err = dsa_destroy(&axisY)) {
   DSA_EXT_DIAG(err, axisY);
   goto _error;
}
```

9 Recommended proper reaction to error detection

Code is more maintainable if all error processing is done in the same location. This is why, in the above examples, function call error handling has always referred to a goto label, where all error processing is performed.

Of course, one drawback of a single common error handler is that it needs to identify where the error was originated and what is the state of the program. This might make more code to write, but once validated, it usually ends up making the software easier to maintain.

The group object of drives, if it is a valid pointer, can first be destroyed; it is no longer of any use. The group, drive, UltimET pointers can be invalid if the `dsa_create_...` functions have failed (or not been called!) or that some memory has been corrupted.

```
if (dsa_is_valid_drive_group(igrp))
   dsa_destroy(&igrp);
```

The same has to be done with the motion controller and the position controllers. For each position controller, if the object pointer is valid, first check if the communicate is still working in order to stop all movements, power off the controller and finally close the communication. Afterwards, the object itself can be destroyed:

```
if(dsa_is_valid_drive(axisX)) {
   /* Is the communication open? */
   ebool open = 0;
   dsa_is_open(axisX, &open);
   if (open) {
      /* Stop movements. */
      dsa_quick_stop_s(axisX, DSA_QS_PROGRAMMED_DEC,
                       DSA_QS_BYPASS | DSA_QS_STOP_SEQUENCE, DSA_DEF_TIMEOUT);

      /* When the motor has stopped, a power off is done. */
      dsa_wait_movement_s(axisX, 60000);
      dsa_power_off_s(axisX, 60000);

      /* Close the connection. */
      dsa_close(axisX);
   }

   /* Finally, release the associated resources to the OS. */
   dsa_destroy(axisX);
}
```

The same must be done for the other axis. The UltimET also has to be shut down:

```
if(dsa_is_valid_master(UltimET)) {
   /* Is the UltimET open ? */
   ebool open = 0;
   dsa_is_open(UltimET, &open);
   if (open) {
      /* Close the connection */
      dsa_close(UltimET);
   }
   /* And finally, release all resources to the OS. */
   dsa_destroy(&UltimET);
}
```

Remark: The dsa_destroy function requires as parameter the address of the pointer to the object (`&axisX`, `&axisY` and `&UltimET`) and not the pointers themselves (`axisX`, `axisY` and `UltimET`). In this way, once the object is destroyed, its address is assigned to NULL.

10 Units and unit conversions

10.1 Specific functions

In this type of function, the parameters representing a physical quantity are always given in ISO units. For example, in the `dsa_set_target_position()`, the linear position is given in meter. According to the motor type (rotary or linear) and the quantity (position, speed, time...), the value will be given with the following unit. By using specific functions, EDI will automatically have access to the correct parameter depending on the family of the accessed device.

Quantity	Linear motor	Rotary motor
Position	m	Turns
Speed	m/s	Turns/s
Acceleration	m/s^2	$Turns/s^2$
Jerk time	s	
Time	s	
Current	A	
Temperature	°C	

10.2 Generic functions

As seen previously, the parameters of the functions representing a physical quantity are always given in ISO unit like meter, second, ampere, etc... On the other hand, ETEL devices work with internal units called «increments». Here is a table illustrating the differences between these two types of units:

	ISO units	Increments
Coding	Floating point 64 bits	32 or 64 bits integer or float
Reference	Normalized	Depending on the device's parameters
Names	meter [m], second [s], ampere [A],...	UPI, USI, DPI,...
Use	Application level	Device level

To set a register in increment using the generic functions, the user needs to know the register-type and register-index in order to use the adequate function:

	DSA function
Set register containing 32-bit integer increments	dsa_set_register_int32_...
Set register containing 64-bit integer increments	dsa_set _register_int64_...
Set register containing 32-bit float increments	dsa_set_register_float32_...
Set register containing 64-bit float increments	dsa_set_register_float64_...

To set a register in ISO using the generic functions, EDI will automatically detect the accessed register and convert ISO double parameter into the correct increment type.

	DSA function
Set register using ISO value	dsa_set_iso_register_...

When using the specific functions, the DSA library does automatically the necessary unit conversions and send to the devices the quantities in increments. On the other hand, when the generic functions are used, the user can choose between the use of increments or ISO units. If the ISO unit is chosen, the user must specify the name of corresponding unit in the device so that the DSA library does the appropriate conversion.

Here are some examples:

Example 1:

- The user wants to modify the K39 register of a position controller
- K39 represents the in-window position of a linear motor
- This quantity in increments (in the position controller) is given in UPI
- This quantity in ISO units is given in meters [m]

The allocation of this register in increments is done in the following way:

```
err = dsa_set_register_int32_s(
    drive1,                    /* grp: destination device */
    2,                         /* typ: 2 = K register */
    39,                        /* idx: register index */
    0,                         /* sidx: register sub-index */
    1000,                      /* value: register value in UPI */
    DSA_DEF_TIMEOUT            /* timeout: default timeout */
);
```

or

To set a register in increment using the generic functions, the user needs to know the corresponding register-type and register-index to use the adequate function:

```
err = dsa_set_register_int32_s(
   drive1,                     /* destination device*/
   DMD_TYP_PPK,                /* typ 2 = K register */
   39,                         /* register index */
   0,                          /* register sub-index*/
   1000,                       /* value: register value in UPI*/
   DSA_DEF_TIMEOUT             /* default timeout*/
);
```

The appointment of this register in ISO units is done in the following way:

```
err = dsa_set_iso_register_s(
    drive1,                    /* grp: destination device */
    2,                         /* typ: 2 = K register */
    39,                        /* idx: register index */
    0,                         /* sidx: register sub-index */
    0.0001,                    /* value: register value in [m] */
    DMD_CONV_UPI,              /* conv: drive unit (UPI) */
    DSA_DEF_TIMEOUT            /* timeout: default timeout */
);
```

To set a register specifying the ISO value, the user must specify which conversion function number to use. Here follows a list of available conversion function number:

Values (#define)	Device's unit	Description	Condition
DMD_CONV_AVI	AVI	Analog voltage: +/-8192 inc => -/+10V	Deprecated
DMD_CONV_AVI12BIT	AVI12BIT	Analog voltage: +/-2048 inc => -/+10V	Deprecated
DMD_CONV_AVI16BIT	AVI16BIT	Analog voltage: +/-32767 inc => -/+10V	Deprecated
DMD_CONV_AVI16BITINV	AVI16BITINV	Analog voltage: +/-32767 inc => +/-10V	
DMD_CONV_BIT0	BIT0	2^0 = 1 correspond to 1.0	Deprecated
DMD_CONV_BIT10	BIT10	2^10 = 1024 correspond to 1.0	Deprecated
DMD_CONV_BIT11	BIT11	2^11 = 2048 correspond to 1.0	Deprecated
DMD_CONV_BIT11_ENCODER	BIT11_ENCODER	Analog encoder signal amplitude in volt (11 bit)	
DMD_CONV_BIT11P2	BIT11P2	Analog encoder (11 bit)	

Values (#define)	Device's unit	Description	Condition
DMD_CONV_BIT15	BIT15	2^15 = 32768 correspond to 1.0	
DMD_CONV_BIT15_ENCODER	BIT15_ENCODER	Analog encoder signal amplitude in volt (15 bit)	
DMD_CONV_BIT15P2	BIT15P2	Analog encoder (15 bits)	Deprecated
DMD_CONV_BIT24	BIT24	2^24 = 256*65536 correspond to 1.0	Deprecated
DMD_CONV_BIT31	BIT31	2^31 = 32768*65536 correspond to 1.0	Deprecated
DMD_CONV_BIT5	BIT5	2^5 = 32 correspond to 1.0	Deprecated
DMD_CONV_BIT8	BIT8	2^8 = 256 correspond to 1.0	Deprecated
DMD_CONV_BIT9	BIT9	2^9 = 512 correspond to 1.0	Deprecated
DMD_CONV_BOOL	BOOL	Boolean value	
DMD_CONV_C13	C13	Current 13bit range	Deprecated
DMD_CONV_C14	C14	Current 14bit range	Deprecated
DMD_CONV_C29	C29	Current 29bit range	
DMD_CONV_CLTI	CLTI	Current loop time increment (cti)	Deprecated
DMD_CONV_CTI	CTI	Current loop time increment (41us)	Deprecated
DMD_CONV_CTRL_CUR2	CTRL_CUR2	Controller i^2*t, dissipation value	
DMD_CONV_CTRL_CUR2T	CTRL_CUR2T	Controller i^2*t, integration value	
DMD_CONV_CUR	CUR	Current	
DMD_CONV_CUR_NM	CUR_NM	Current in Newton meter	Deprecated
DMD_CONV_CUR2	CUR2	i^2, dissipation value	
DMD_CONV_CUR2PHYSICAL	CUR2PHYSICAL	Cur2 using M82:1	
DMD_CONV_CUR2T	CUR2T	i^2*t, integration value	
DMD_CONV_CUR2T_V2	CUR2T_V2	i^2*t, integration value	
DMD_CONV_CUR2TPHYSICAL	CUR2TPHYSICAL	cur2t using M82:1	
DMD_CONV_CURPHYSICAL	CURPHYSICAL	dci factor using M82:1	
DMD_CONV_DAI	DAI	Drive acceleration increment	
DMD_CONV_DPI	DPI	Drive position increment	
DMD_CONV_DPI2	DPI2	Drive position increment (secondary encoder)	
DMD_CONV_DPI3	DPI3	Drive position increment (auxiliary encoder)	
DMD_CONV_DPIPHYSICAL	DPIPHYSICAL	dpi without Mimo conversion (MF500)	Deprecated
DMD_CONV_DSI	DSI	Drive speed increment	
DMD_CONV_DWORD	DWORD	Double word value without conversion	
DMD_CONV_ENCOFF	ENCOFF	11bit with 2048 offset	Deprecated
DMD_CONV_EXP10	EXP10	Ten power factor	Deprecated
DMD_CONV_FLOAT	FLOAT	Float value	
DMD_CONV_FREF_FCTRL	FREF_FCTRL	Force ref of force control (conv 1:1)	
DMD_CONV_FTI	FTI	Fast time increment (125us-166us)	
DMD_CONV_HSTI	HSTI	Half slow time increment	Deprecated
DMD_CONV_INT	INT	Integer value without conversion	
DMD_CONV_IP_ADDRESS	IP_ADDRESS	ip address type	
DMD_CONV_ISO_POS	ISO_POS	Increment register in ISO (position) (m or t 1:1)	
DMD_CONV_K1	K1	Position loop proportional gain	
DMD_CONV_K10	K10	1st order filter in s.	Deprecated
DMD_CONV_K1031	K1031	Per cent unit, 100% => 3133	Deprecated
DMD_CONV_K14	K14	Dual encoder speed loop output (DSC)	Deprecated
DMD_CONV_K2	K2	Position loop speed feedback gain	
DMD_CONV_K20	K20	Position loop speed ffwd gain	
DMD_CONV_K20_DSB	K20_DSB	Position loop speed ffwd gain (DSB)	Deprecated
DMD_CONV_K21	K21	Position loop acceleration ffwd gain	
DMD_CONV_K21_DSB	K21_DSB	Position loop acceleration ffwd gain (DSB)	Deprecated
DMD_CONV_K23	K23	Commutation phase advance factor	
DMD_CONV_K23_ACCURET	K23_ACCURET	Back EMF compensation	

Values (#define)	Device's unit	Description	Condition
DMD_CONV_K239	K239	motor Kt factor in mN(m)/A	
DMD_CONV_K4	K4	Position loop integrator gain	
DMD_CONV_K5	K5	Position loop anti-windup gain	
DMD_CONV_K75	K75	Encoder multiple index distance	
DMD_CONV_K8	K8	Position loop speed filter (DSC)	Deprecated
DMD_CONV_K80	K80	cl prop gain delta[1/A]	
DMD_CONV_K80_VHP	K80_VHP	cl prop gain delta[V/A]	
DMD_CONV_K80_VHPPHYSICAL	K80_VHPPHYSICAL	K80_VHP using M82:1	
DMD_CONV_K80PHYSICAL	K80PHYSICAL	K80 using M82:1	Deprecated
DMD_CONV_K81	K81	cl prop integrator delta[1/(A*s)]	
DMD_CONV_K81_VHP	K81_VHP	cl prop gain delta[V/(A*s)]	
DMD_CONV_K81_VHPPHYSICAL	K81_VHPPHYSICAL	K81_VHP using M82:1	
DMD_CONV_K81PHYSICAL	K81PHYSICAL	K81 using M81:1	Deprecated
DMD_CONV_K82	K82	Filter time, T = [cti] * ((2^n)-1)	Deprecated
DMD_CONV_K9	K9	1st order filter in pl	Deprecated
DMD_CONV_K94	K94	Time in 2x current loop increment	Deprecated
DMD_CONV_K95	K95	Current rate for k95	Deprecated
DMD_CONV_K96	K96	Phase rate for k96	Deprecated
DMD_CONV_KDF_FCTRL	KDF_FCTRL	Force loop derivative gain	
DMD_CONV_KF22	KF22	Jerk feedforward	
DMD_CONV_KF256	KF256	Kt/M for Init small movement 2	
DMD_CONV_KFLOAT	KFLOAT	Float value for K parameters	
DMD_CONV_KIF_FCTRL	KIF_FCTRL	Integrator gain for the force loop	
DMD_CONV_KPF_FCTRL	KPF_FCTRL	Force loop proportional gain	
DMD_CONV_KT_MOTOR	KT_MOTOR	KT motor	
DMD_CONV_LONG	LONG	Long integer value without conversion	
DMD_CONV_M16	M16	Jerk value	
DMD_CONV_M242	M242	Quartz frequency in Hz	
DMD_CONV_M29	M29	Per cent unit, 100% => M229	
DMD_CONV_M82	M82	Current limit in 10 mA units	
DMD_CONV_MASS_ACC_FFWD	MASS_ACC_FFWD	Mass for acceleration feed forward	Deprecated
DMD_CONV_MASS_ACC_FFWD2	MASS_ACC_FFWD2	Mass for acceleration feed forward	
DMD_CONV_MF89	MF89	Magnetic period for Init small movement 2	
DMD_CONV_MLTI	MLTI	Manager loop time increment (mlti)	
DMD_CONV_MSEC	MSEC	Milliseconds	
DMD_CONV_PER_100	PER_100	Per cent unit, 100% => 1.0	
DMD_CONV_PER_1000	PER_1000	Per thousand unit	
DMD_CONV_PH11	PH11	2^11 = 2048 correspond to 360 degrees	
DMD_CONV_PH12	PH12	2^12 = 4096 correspond to 360 degrees	Deprecated
DMD_CONV_PH28	PH28	2^28 = 65536*4096 correspond to 360 degrees	Deprecated
DMD_CONV_PLTI	PLTI	Position loop time increment (fti)	
DMD_CONV_PLTI_INV	PLTI_INV	1/plti (1/M244)	
DMD_CONV_POLE_FREQ	POLE_FREQ	Filter pole frequency in Herz	
DMD_CONV_QZTIME	QZTIME	Interrupt time in sec = inc / m242	
DMD_CONV_SEC	SEC	Seconds (1:1)	
DMD_CONV_SPEC2F	SPEC2F	Filter time, T = [fti] * ((2^n)-1)	Deprecated
DMD_CONV_STI	STI	Slow time increment (500us-2ms)	
DMD_CONV_STRING	STRING	Packed string value	
DMD_CONV_TEMP	TEMP	2^0 = 1 correspond to 1.0	
DMD_CONV_TTI	TTI	Minimum time base TransnET (25us)	
DMD_CONV_UAI	UAI	Acceleration, user acceleration increment	

Values (#define)	Device's unit	Description	Condition
DMD_CONV_UFAI	UFAI	User friendly acceleration increment	
DMD_CONV_UFPI	UFPI	User friendly position increment	
DMD_CONV_UFSI	UFSI	User friendly speed increment	
DMD_CONV_UFTI	UFTI	User friendly time increment	
DMD_CONV_UPI	UPI	User position increment	
DMD_CONV_UPI2	UPI2	User position increment (dual encoder)	
DMD_CONV_UPI3	UPI3	User position increment (auxiliary encoder)	Deprecated
DMD_CONV_USI	USI	User speed increment	
DMD_CONV_VOLT	VOLT	2^0 = 1 correspond to 1.0	
DMD_CONV_VOLT100	VOLT100	(2^0)/100 = 1 correspond to 1.0	

The specific functions request ISO parameters values. However, in certain cases, the user might be interested in converting increments to ISO or vice-versa. For these cases, EDI provides some functions allowing to convert increments to/from ISO:

```
dsa_convert_int32_to_iso(DSA_DEVICE *dev, double *iso, long inc, int conv);
dsa_convert_int64_to_iso(DSA_DEVICE *dev, double *iso, eint64 inc, int conv);
dsa_convert_float32_to_iso(DSA_DEVICE *dev, double *iso, float inc, int conv);
dsa_convert_float64_to_iso(DSA_DEVICE *dev, double *iso, double inc, int conv);
dsa_convert_int32_from_iso(DSA_DEVICE *dev, long *inc, double iso, int conv);
dsa_convert_int64_from_iso(DSA_DEVICE *dev, eint64 *inc, double iso, int conv);
dsa_convert_float32_from_iso(DSA_DEVICE *dev, float *inc, double iso, int
                             conv);
dsa_convert_float64_from_iso(DSA_DEVICE *dev, double *inc, double iso, int
                             conv);
```

The conv parameter represents the conversion which must be used. The corresponding conversion of a register can be retrieved using:

- the DMD function: `dmd_get_register_convert(DMD *dmd, int typ, unsigned idx, int sidx)`
- the DSA macro: `DSA_REG_CONV(registerTyp, registerIndex, registerSidx);`

Some conversions are not considered as ISO conversion. Then, before calling the conversion function, it is advised to check if the conversion is an ISO conversion using the function `dmd_is_double_conv(int conv)`. Otherwise, calling the conversion function will return a `DSA_EBADPARAM ERROR`.

Here is the list of the conversion functions which are not ISO conversion:

```
DMD_CONV_DWORD
DMD_CONV_BOOL
DMD_CONV_INT
DMD_CONV_LONG
DMD_CONV_STRING
DMD_CONV_FLOAT
DMD_CONV_ADDRESS
```

Almost all ISO conversions can be represented by a single factor. Then, the register increments can easily be converted into ISO using this factor.

The DSA function `dsa_get_inc_to_iso_factor(DSA_DEVICE *dev, int conv, double *factor)` allows to retrieve this factor. However, some ISO conversions **cannot** be represented by a factor. In this case, calling `dsa_get_inc_to_iso_factor` function will return a `DSA_ECONVERT` error.

Here is the list of the ISO conversions which cannot be represented by a factor:

`DMD_CONV_KFLOAT`	(K256, K257, K258, k259, k260, k261, K262, k263, k264, K265, K266, K267, K268, MF92, MF93, MF94
`DMD_CONV_POLE_FREQ`	(KF12)
`DMD_CONV_EXP10`	(Obsolete)
`DMD_CONV_SPEC2F`	(Obsolete)

DMD_CONV_ENCOFF	(Obsolete)
DMD_CONV_BIT11P2	(M43)
DMD_CONV_BIT15P2	(Obsolete)
DMD_CONV_K82	(Obsolete)

Therefore, for these ISO conversions, the EDI functions "dsa_convert_..._to/from_iso" **must** be used if the user wants to do the conversions from increments to ISO.

To use the #define's values, the 'dmd40.h' file must be included at the beginning of the source file as follows:

```
#include <dmd40.h>
```

The unit of a parameter is then represented by an integer assigned to the 'conv' parameter of the generic functions. This number represents the ID of the unit in question.

To know the unit of a register or the unit of a command's parameter, that is to say the value to give to the 'conv' parameter, three methods are available:

- Check in the "Operation & Software manual" of the device in question, the 'units conversion' chapter. Thanks to the name of the unit given in this chapter and the defines written above, the value of the 'conv' parameter can be found.
- Use the dsa_get_register_...() function. If the 'kind' parameter is assigned to DSA_GET_CONV_FACTOR, this function returns in the 'val' parameter the unit of the register. This method cannot be used to know the unit of a command's parameters.
- Use the DSA_REG_CONV(typ, idx, sidx) macro which is replaced by the unit of the register specified in the parameters. The DSA_CMD_CONV(typ, idx, par) macro allows the same thing for the parameters of the commands. The values given by these macros are different from the #define ones or those given by dsa_get_register_...(). Their use and the result are identical.

10.3 System registers

System registers are the registers influencing ISO conversion factors. These registers are read and internally stored by EDI when the communication is opened. The ISO conversion functions will then use these internally stored value to fulfill ISO conversion. If the user modifies the value of a system register, then EDI must be forced to reload the system registers by calling the EDI dsa_reset() function.

Here is the list of system registers:

Registers	Aliases	Description
C1		Manager mode
C8		Position loop time interrupt setting
C9		Manager loop time interrupt setting
C490		ZxT transfer matrix selection
K54	PPOLE	Motor pair pole number
K55		Encoder position increment factor
K72		TTL filter length
K76		Dual Encoder Feedback mode
K77		Encoder interpolation shift value
K78		Regulator type
K79		Encoder type
K88		Current regulator mode
K89		Motor phase number and PWM frequency
K118		Encoder HSEI input filter
K239		Motor Kt factor
K240		Movement type conversion
K241		Encoder period
K242		Position multiplication factor
K243		Position division factor

Registers	Aliases	Description
K522		Ufpi multiplication factor
K523		Ufpi ten power
K524		Ufsi ten power
K525		Ufai ten power
K526		Ufti ten power
KF490		ZXT Conversion radius
KL55		Encoder position increment factor
M81		SAV and Reboot needed for ISO conversion
M82		Controller max current
M229		Max value of PWM counter
M239		Encoder period
M240		Movement type conversion
M241		Encoder interpolation factor
M242		UltimET Light quartz frequency [Hz]
M243	CLTI	Controller current loop time factor
M244	ULTI	UltimET Light interrupt time factor
M245	MLTI	Controller manager loop time factor
MF82	IMAXC	Controller max current
MF229		Output voltage conversion factor
MF500		Logical axes MIMO correction factor
MF510		ZXT FFW mass iso conversion

11 Access to the controller parameters

Access to controller parameters

11.1 Sending commands

Some commands do not have a corresponding specific function and it is necessary to use the generic functions to send them to the controller. The user must pay attention to the fact that these functions are for Advanced and non-standard use (e.g. indirect addressing) and could lead to unexpected behavior. For further information, please contact ETEL's Support team.

Each device is able to execute a large number of commands which do all sorts of operations.

Each command has a number and none, one or several parameters. These parameters can belong to one of the following categories:

- Parameters with units: they represent physical quantities such as position, speed, time, etc. They can be given in increments or ISO units. In increments, they are coded in 32-bit or 64-bit integer or float values. In ISO units, they are coded in double precision floating point values.
- Parameters without units: they often represent non-physical quantities such as the axis number, the error number, a digital input/output state, etc... These quantities are coded in 32-bit integers.
- Special parameters: they represent position controller registers. This kind of parameters is not further developed in this "User's Manual".

Here are examples of commands that a position controller is able to execute:

Command	ETEL syntax	Command number	Parameter number	Parameters type
Power on	PWR	124	1	Without unit
Emergency stop	HLO	119	0	Without unit
Set axis number	AXI	109	2	Without unit
Waiting time	WTT	10	1	With unit (physical quantity)
Reset	RSD	88	1	Without unit

The DSA library has a set of functions which allows the user to send a command to a device. Among these functions, the user will choose one according to the number and the type of parameters of the command to send.

As commands can have several types of parameters, EDI defines only a subset of generic functions.

Here is the list of the available functions:

Function which sends the command	Parameters type of the command to send
dsa_execute_command_s()	No parameter
dsa_execute_command_d_s()	One 32-bit integer parameter
dsa_execute_command_i_s()	One ISO parameter
dsa_execute_command_dd_s()	Two 32-bit integer parameters
dsa_execute_command_id_s()	Two parameters: the first in ISO and the second in 32-bit integer
dsa_execute_command_di_s()	Two parameters: the first in 32-bit integer and the second in ISO
dsa_execute_command_ii_s()	Two parameters in ISO
dsa_execute_command_x_s()	Any number and type of parameters

If the user wants to send the command number 119 (HLO) to a position controller, the function to use is `dsa_execute_command_s()` as follows:

```
err = dsa_execute_command_s(
   axisX,                    /* grp: destination device */
   119                       /* cmd: number of the command */
   FALSE,                    /* fast: fast command */
```

```
   FALSE,                    /* ereport: report drive errors */
   DSA_DEF_TIMEOUT           /* timeout: by default */
);
```

In the same way, here is an example to send the command number 109 (AXI) with the parameters 1203 and 3:

```
err = dsa_execute_command_dd_s(
   axisX,                    /* grp: destination device */
   109,                      /* cmd: number of the command */
   0,                        /* typ1: type of the first parameter */
   1203,                     /* par1: first parameter */
   0,                        /* typ2: type of the second parameter */
   3,                        /* par2: second parameter */
   FALSE,                    /* fast: fast command */
   FALSE,                    /* ereport: report drive errors */
   DSA_DEF_TIMEOUT           /* timeout: by default */
);
```

Regarding the meaning of the different parameters, the first one is always the `DSA_DRIVE` or `DSA_MASTER` object which represents the device on which the operation will be executed. Among the last parameters, there is the 'timeout' with two other parameters. The remaining parameters depend on the number and the type of the command's parameters to send.

The following table shows all the parameters:

Parameter	Name	Description
First parameter	grp	DSA_DRIVE or DSA_MASTER object
Parameter dependent on the number and the type of the command's parameters to send	typ	Always equal to 0 which means that the parameter is an immediate value. During the use of commands with special parameters, this parameter can have other values.
	par	Value of the parameter. It can be a long or a double type depending on whether the value is given in increments or ISO units.
	conv	Type of conversion to do. This parameter is only present when the parameter is used with the value given in ISO unit. See §10.1.
Last parameters	fast	Equal to TRUE if the command is fast. The fast commands have priority on the others. Only a few commands can be a fast command. In the majority of cases, this parameter must be equal to FALSE.
	ereport	If this parameter is equal to TRUE and the device which the command is sent to is in error, the function returns an error. It is then possible to detect if a device is in error or not.
	timeout	Maximum time of execution.

The parameters of the `dsa_execute_command_x_s()` function are a little bit different. Actually, this function allows the user to send commands whose number of parameters is variable and/or type are different. First, the user must create a `DSA_COMMAND_PARAM` type array with a number of elements corresponding to the number of the command's parameters. Each element in that table must then be assigned with the type, the value and the conversion type of the corresponding parameter. Once the array is created and assigned, the `dsa_execute_command_x_s()` command must be called by passing the pointer and size of this array.

Here is an example where 1025 command (*ILINE=0, 0.28, 0.12, 0, 0) is sent to the UltimET:

```
DSA_COMMAND_PARAM params[ ] = {
   {0,0,0},{0,0,0},{0,0,0},{0,0,0},{0,0,0}
};

params[1].conv = DMD_CONV_UFPI;
params[1].val.d = 0.28;
params[2].conv = DMD_CONV_UFPI;
params[2].val.d = 0.12;

dsa_execute_command_x_s(UltimET, 1025, params, 5,
                        FALSE, FALSE, DSA_DEF_TIMEOUT);
```

Here is an example where command 61 (SMP command) (SMP.0=0, 12000L, 5000L, 0.05) is sent to a controller:

```
SMP.0 = 0, 12000L, 5000L, 0.05 sub-index, position in 64bits integer, speed in
64-bit integer, acc in ISO

DSA_COMMAND_PARAM param[4];
param[0].typ = DMD_TYP_IMMEDIATE_INT32;
param[0].conv = 0;
param[0].val.i = 0;

param[1].typ = DMD_TYP_IMMEDIATE_INT64;
param[1].conv = 0;
param[1].val.i64 = 12000;

param[2].typ = DMD_TYP_IMMEDIATE_INT64;
param[2].conv = 0;
param[2].val.i64 = 5000;

param[3].typ = DMD_TYP_IMMEDIATE_INT64;
param[3].conv = DMD_CONV_UAI;
param[1].val.d = 0.05;

dsa_execute_command_x(drive1, 61, params, 4, FALSE, FALSE, DSA_DEF_TIMEOUT);
```

11.2 Reading and writing of controller registers

Each device has several registers' types. The 16 types usually called are:

Type	Name	Type number
32-bits integer Parameters	K0, K1, K2,…	DMD_TYP_PPK_INT32 (2) or DMD_TYP_PPK
32-bits float Parameters	KF0, KF1, KF2,…	DMD_TYP_PPK_FLOAT32 (34)
64-bits integer Parameters	KL0, KL1, KL2,…	DMD_TYP_PPK_INT64 (66)
64-bits float Parameters	KD0, KD1, KD2,…	DMD_TYP_PPK_FLOAT64 (98)
32-bits integer Monitoring	M0, M1, M2,…	DMD_TYP_MONITOR_INT32 (3) or DMD_TYP_MONITOR
32-bits float Monitoring	MF0, MF1, MF2,…	DMD_TYP_MONITOR_FLOAT32 (35)
64-bits integer Monitoring	ML0, ML1, ML2,…	DMD_TYP_MONITOR_INT64 (67)
64-bits float Monitoring	MD0, MD1, MD2,…	DMD_TYP_MONITOR_FLOAT64 (99)
32-bits integer Common	C0, C1, C2,…	DMD_TYP_COMMON_INT32 (13) or DMD_TYP_COMMON
32-bits float Common	CF0, CF1, CF2,…	DMD_TYP_COMMON_FLOAT32 (45)
64-bits integer Common	CL0, CL1, CL2,…	DMD_TYP_COMMON_INT64 (77)
64-bits float Common	CD0, CD1, CD2,…	DMD_TYP_COMMON_FLOAT64 (109)
32-bits integer User	X0, X1, X2,…	DMD_TYP_USER_INT32 (1) or DMD_TYP_USER
32-bits float User	XF0, XF1, XF2,…	DMD_TYP_USER_FLOAT32 (33)
64-bits integer User	XL0, XL1, XL2,…	DMD_TYP_USER_INT64 (65)
64-bits float User	XD0, XD1, XD2,…	DMD_TYP_USER_FLOAT64 (97)
32-bits integer ESM(*)	W0,W1,W2,…	DMD_TYP_ESM_INT32(12) or DMD_TYP_ESM
32-bits float ESM(*)	WF0,WF1,WF2,…	DMD_TYP_ESM_FLOAT32(44)
(64-bits integer ESM(*)	WL0,WL1,WL2,…	DMD_TYP_ESM_INT64(76)
64-bits float ESM(*)	WD0,WD1,WD2,…	DMD_TYP_ESM_FLOAT64(108)
32-bits integer ESM Monitor(*)	R0,R1,R2,…	DMD_TYP_ESM_MON_INT32(10) or DMD_TYP_ESM_MON
32-bits float ESM Monitor(*)	RF0,RF1,RF2,…	DMD_TYP_ESM_MON_FLOAT32(42)
64-bits integer ESM Monitor(*)	RL0,RL1,RL2,…	DMD_TYP_ESM_MON_INT64(74)
64-bits float ESM Monitor(*)	RD0,RD1,RD2,…	DMD_TYP_ESM_MON_FLOAT64(106)

(*) These registers are available on the ULTIMET ADVANCED motion controller with an ESM module. Refer to the "ULTIMET ADVANCED Motion Controller Operation and Software Manual" for further details.

The DSA library has a specific function to read and assign each register widely used. Furthermore, it offers 12 generic functions which are able to assign and read any register:

Function	Description
dsa_get_register_int32_s()	Allows the reading of a integer 32-bits increment value of a register of corresponding increment type
dsa_get_register_float32_s()	Allows the reading of a float 32-bits increment value of a register of corresponding increment type
dsa_get_register_int64_s()	Allows the reading of a integer 64-bits increment value of a register of corresponding increment type
dsa_get_register_float64_s()	Allows the reading of a float 64-bits increment value of a register of corresponding increment type
dsa_get_iso_register_s()	Allows the reading of an ISO unit value of any register
dsa_set_register_int32_s()	Allows the setting of a integer 32-bits increment value of a register of corresponding increment type
dsa_set_register_float32_s()	Allows the setting of a float 32-bits increment value of a register of corresponding increment type
dsa_set_register_int64_s()	Allows the setting of a integer 64-bits increment value of a register of corresponding increment type
dsa_set_register_float64_s()	Allows the setting of a float 64-bits increment value of a register of corresponding increment type
dsa_set_iso_register_s()	Allows the setting of an ISO unit value of any register

The following example assigns the X2:0 user variable with the value read in the M64:0 monitoring register:

```
long val;
err = dsa_get_register_int32_s(
   axisX,                    /* grp: destination device */
   3,                        /* typ: 3 = M monitoring register */
   64,                       /* idx: register index */
   0,                        /* sidx: register subindex */
   &val,                     /* value: register value */
   DSA_GET_CURRENT,          /* kind: actual device value */
   DSA_DEF_TIMEOUT           /* timeout: default timeout */
   );
err = dsa_set_register_int32_s(
   axisX,                    /* grp: destination device */
   1,                        /* typ: 1 = X user variable */
   2,                        /* idx: register index */
   0,                        /* sidx: register subindex */
   val,                      /* value: register value */
   DSA_DEF_TIMEOUT           /* timeout: default timeout */
   );
```

The following example is similar to the previous one but with ISO quantities. It reads the speed stored in the KL211:0 parameter, multiplies it by 10 and stores it again in the KL211:0 parameter. The KL211:0 parameter is in USI unit.

```
double speed; /* in m/s */
err = dsa_get_iso_register_s(
   axisX,                    /* grp: destination device */
   66,                       /* typ: 66 = KL parameter */
   211,                      /* idx: register index */
   0,                        /* sidx: register subindex */
   &speed,                   /* value: returned value */
   DMD_CONV_USI,             /* conv: drive unit (USI) */
   DSA_GET_CURRENT,          /* kind: actual device value */
   DSA_DEF_TIMEOUT           /* timeout: default timeout */
  );
err = dsa_set_iso_register_s(
   axisX,                    /* grp: destination device */
   66,                       /* typ: 66 = KL parameter */
   211,                      /* idx: register index */
   0,                        /* sidx: register subindex */
   speed * 10.0,             /* value: value to set */
   DMD_CONV_USI,             /* conv: drive unit (USI) */
   DSA_DEF_TIMEOUT           /* timeout: default timeout */
   );
```

11.3 Saving and resetting

The values of parameters that have been modified are not automatically save to the controller's flash (non-volatile) memory. This means that if a controller is reset, the parameters revert back to the values actually stored in the flash memory and all modifications are lost. To save the modified parameters to the controller's flash memory, the `dsa_save_parameters_s()` function must be executed (it is the equivalent of the SAV command).

Sending the SAV command to a device will cause it to temporarily loose communication and disappear from the network, just reappear later in error. It is possible that the device itself does not have time to acknowledge the SAV command before the communication is lost. Therefore, for this specific case, it is highly recommended not to verify the error code returned by the `dsa_save_parameters_s()` function, as it might be misleading.

The second parameter corresponds to one of the following constants:

dsa_save_parameters_s() function 2nd parameter	Description
DSA_PARAM_SAVE_ALL	Save all information to the flash memory.
DSA_PARAM_SAVE_SEQ_LKT	Save Sequence and user look-up tables to the flash memory.
DSA_PARAM_SAVE_X_PARAMS	Save user (X) registers and parameters (K) to the flash memory.
DSA_PARAM_SAVE_K_C_E_X_PARAMS(*)	Save K, KL, KF, KD, C, CL, CF, CD, EL, X, XL, XF, XD parameters to flash memory.
DSA_PARAM_SAVE_K_PARAMS(*)	Save K, KL, KF, KD parameters to the flash memory.
DSA_PARAM_SAVE_C_PARAMS(*)	Save C, CL, CF, CD parameters to the flash memory.
DSA_PARAM_SAVE_X_PARAMS(*)	Save X, XL, XF, XD parameters to the flash memory.
DSA_PARAM_SAVE_L_PARAMS(*)	Save LD parameters to the flash memory.
DSA_PARAM_SAVE_SEQUENCES(*)	Save Sequences to the flash memory.
DSA_PARAM_SAVE_K_E_PARAMS(*)	Save K, KL, KF, KD, EL parameters to the flash memory.
DSA_PARAMS_SAVE_P_PARAMS(*)	Save P parameters to the flash memory.
DSA_PARAMS_SAVE_W_PARAM(**)	Save W parameters to the micro-SD memory.

(*) Only on the AccurET family.
(**) Only on the ULTIMET ADVANCED with an ESM module.

As mentioned before, the SAV command will cause the device to temporarily disappear from the network and reappear in error. So, before issuing new commands, it is required to wait enough time until the device is present again:

```
{
   /* setup status masks */
      DSA_STATUS status_checkbits   = {sizeof(DSA_STATUS)}
      DSA_STATUS status_checkstates = {sizeof(DSA_STATUS)};

   status_checkbits.drive.present   = 1;
   status_checkstates.drive.present = 1;

   /* wait that the controllers are present */
   if (err = dsa_wait_status_equal_s(axisX, &status_checkbits,
                                     &status_checkstates, NULL, 20000)) {
      DSA_EXT_DIAG(err, axisX);
      goto _error;
   }
}
```

Remark: The SAV command takes quite some time to execute, hence the large timeout value for the last parameter of the function `dsa_wait_status_equal_s` in the example above.

For the new values of the modified parameters to be effectively taken into account, the controller must be reset. There is no specific function for the RSD (reset) command, so it must be sent with a generic function:

```
dsa_execute_command_d_s(
   axisX,                    /* grp: destination device */
   DMD_CMD_RESET_DRIVE,      /* cmd: constant number of the RSD command (88) */
   DMD_TYP_IMMEDIATE,        /* typ: the parameter to the command is a value*/
   255,                      /* par: the value always passed to RSD command*/
   FALSE,                    /* fast: 1 or true = fast command */
   FALSE,                    /* ereport: 1 or false do not report drive error */
   10000                     /* timeout: 10 secs */
 )
```

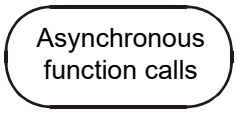
Remark: The RSD command also takes quite some time to be executed, hence the large timeout value for the last parameter of the function `dsa_execute_command_d_s` in the example above.

Also, because the controller resets, it does not have the time to send back the acknowledgement that the reset command was received and executed. This is why the call to `dsa_execute_command_d_s()` for the RSD command returns most of the time an error indicating an absence of command acknowledge. As for the SAV command, it is also recommended not to verify the error code returned by the RSD command, as it might be misleading.

The user must pay attention to the fact that if a motion controller is in the group of devices that is reset, the communication and the connection to that device (and hence all controllers downstream) will be lost. In that case, the communication must be re-established. The correct procedure is to first close the communication with all devices and then re-open it as follows:

```
/* Close communication.
* You don't need to destroy the objects; these will be used again when
* reopening the communication
*/
dsa_close(axisX);
dsa_close(axisY);
dsa_close(UltimET);

/* Reopen communication*/
if (err = dsa_open_u(axisX, "etb:UltimET:0")) {
   DSA_EXT_DIAG(err, axisX);
   goto _error;
}
if (err = dsa_open_u(axisY, "etb:UltimET:1")) {
   DSA_EXT_DIAG(err, axisY);
   goto _error;
}
if (err = dsa_open_u(UltimET, "etb:UltimET:*")) {
   DSA_EXT_DIAG(err, UltimET);
   goto _error;
}
```

12 Asynchronous function calls



This section illustrates how to use asynchronous functions. Before closing the communication and ending this example, the power will be switched back on once again, but this time asynchronously (which means the program will not at this point wait for the power on to finish). This is done using a similar function as at the beginning, but tending with **_a**. As explained before (refer to §3.4.2), asynchronous functions have the following form:

`operation_name`**`_a`**`(`*`device/group identifier, callback function, callback function user parameters`*`).`

So, in the present case, the call would look like this:

```
if (err = dsa_power_on_a(igrp, callback, (void*)1)) {
  DSA_EXT_DIAG(err, igrp);
  goto _error;
}
```

In the above call, the EDI library will start the power on (PWR.x=1) and return immediately. When the power on command finishes, the EDI library calls the function "callback" that was specified and passes it the value of 1 as user parameter.
The callback function must have the following syntax:

```
void callback(DSA_DEVICE_GROUP *grp, int err, void *param) {
... do something when the command - here "power on" - is finished.
}
```

The first parameter is the usual device or device group identifier.
The second is the error code that was returned by the command that was executed. Here, it is the error code of the "power on" command. Usually the callback function does something different depending on whether there was an error or not. In this example, it prints a different message.

The third parameter is what was specified as the third parameter of the asynchronous function call: "1". Hence, in this example, the callback function looks like this:

```
void callback(DSA_DEVICE_GROUP *grp, int err, void *param)
{
  /* The asynchronous function has terminated with an error */
  if (err) {
    asyncFunctionError = err;
    printf("\n-> callback function called with error %d and parameter %d\n",
    err,(int)(param));
  }
  /* The asynchronous function has terminated without error */
  else {
    asyncFunctionError = 0;
    printf("\n-> callback function called with parameter %d\n",
    (int)(param));
  }
  callbackCalled = 1;
}
```

The callback sets the global variable `asyncFunctionError` as a means to transfer the command error code back to the main program, which can then take the appropriate action. Notice also the last line, which sets a global variable to tell the main program that the function has been called. This is done here to allow displaying "*"s while the callback has not been called as follows:

```
callbackCalled = 0;
if (err = dsa_power_on_a(igrp, callback, (void*)1)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}
while (!callbackCalled) {
   printf("*");
   fflush(stdout);
   Sleep(100);
}
```

Somewhere amongst the "*"s, one of the messages "callback function called with error XXX and parameter 1" or "callback function called with parameter 1" will be displayed, depending on how the power on terminates.
Since there is an asynchronous function for every "normal" one, it is also possible to wait for the end of a movement using an asynchronous function. In the example, this is illustrated with a homing command (IND.x). The homing command is started as usual, but waiting for the end of the associated movement is performed with an asynchronous call:

```
if (err = dsa_homing_start_s(igrp, 10000)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}

/* Initialization of main thread - callback function synchronous variable */
callbackCalled = 0;
if (err = dsa_wait_movement_a(igrp, callback, (void*)2)) {
   DSA_EXT_DIAG(err, igrp);
   goto _error;
}

/* Wait for callback function to be called and finished*/
while (!callbackCalled) {
   printf("*");
   fflush(stdout);
   Sleep(100);
}

if (asyncFunctionError) {
   DSA_EXT_DIAG(asyncFunctionError, igrp);
   goto _error;
}
```

Here amongst the "*"s, one of the messages "callback function called with error XXX and parameter 2" or "callback function called with parameter 2" will be displayed, depending on how the homing terminates. Notice "2" is the difference with the previous message seen for the power on.

Notice also how the error code of the command can be processed as usual error codes using a global variable - `asyncFunctionError`, and the `DSA_EXT_DIAG` function.

13 Acquisition

ETEL's controllers can save information of internal registers under specific conditions which a user application can retrieve for the most various purposes. This whole process is known as acquisition and entails 3 actions:

- **Configuration**: the user application must configure which register(s) to acquire, how many samples at which sampling rate and, finally, when/how to start the acquisition (trigger conditions);
- **Acquisition**: once the trigger conditions are met, the controller starts to save the register's information over time up to the defined number of samples and at the defined sampling frequency;
- **Upload**: once all samples have been saved, the data is ready to be uploaded from the controller to the user application.

Remark: During an acquisition the controller uses the Trace advanced register type (T register) to store the values of the basic register types K, M, X and C over time. A trace corresponds to a T register containing the values of the register selected for acquisition, sampled at the frequency set by the user and with the acquisition starting from the moment the trigger conditions, also set by the user, are valid. Refer to the "AccurET Modular Position Controller Operation & Software Manual" and the "UltimET Light / ULTIMET ADVANCED Motion Controller User's Manuals" for more information about register types.

ETEL's controllers support two acquisition modes:

- **Normal**: after configuring the acquisition, the user application instructs the controller to start the acquisition, waits for the controller to finish acquiring the data (trigger conditions must be valid and all points saved) and finally uploads the acquisition data. For acquiring more data, a new acquisition must be programmed.
- **Continuous**: for this mode there is no need for configuring the trigger conditions. As soon as the user application instructs the controller to start the acquisition, the data is continuously saved to a circular buffer. The user application must upload the traces fast enough from the controller to avoid overrunning already acquired samples (data loss). A few guidelines are provided in §13.3 for configuring correctly the continuous acquisition to avoid data loss (samples overrun on the circular buffer). In this mode, the acquisition is performed continuously until stopped without the need for reprogramming.

13.1 Acquisition synchronization

When configuring a single acquisition for retrieving data from multiple AccurET position controllers connected to the same TransnET bus (slaves), the acquisition is synchronized by the UltimET motion controller (master). It is not possible to realize a single acquisition to retrieve data from multiple AccurET position controllers that are not interconnect through the same TransnET bus. For such case, the user must configure multiple acquisitions, one for each controller. Also, for such case, as each controller has its own time reference because they are not interconnected through the TransnET bus, the acquired data is not synchronized between controllers (the exception being the acquisition on both axes of the same AccurET controller). If an UltimET motion controller is already used to synchronize an acquisition, it is not possible to configure another acquisition using the same UltimET controller as master.

13.2 EDI programming: Normal Acquisition

Remark: For details about the EDI acquisition functions, please refer to the corresponding HTML documentation included in the EDI package.

13.2.1 Creating an acquisition

EDI provides the `DSA_ACQUISITION` object which the user must use for programming an acquisition. When creating this object, the user specifies which devices take part of the acquisition, namely:

- Devices which will acquire data;
- Devices which will act as trigger;
- Devices which will synchronize the acquisition (e.g. UltimET motion controller).

The EDI function for creating an acquisition object uses dynamic memory allocation. The continuous usage of dynamic memory allocation is not recommended because of memory fragmentation and performance degradation, in particular for real-time applications. Thus, it is recommended to create the acquisition objects

altogether when starting the application and destroy them just before exiting the application.

```
int dsa_create_acquisition(DSA_ACQUISITION **acq, DSA_DEVICE_BASE *dev)
```

13.2.2 Reserving an acquisition

An acquisition object must be reserved before it can be used. It is at this particular moment that EDI actually verifies if every device specified at object creation is not already taking part on another reserved acquisition. From a programming point of view, realizing this check earlier when creating the acquisition object would be too constraining considering the recommendation for creating acquisition objects altogether at application startup. With this mechanism of reservation, a user can create all the acquisition objects as recommended and reserve them in a need to use basis.

```
int dsa_acquisition_reserve(DSA_ACQUISITION *acq)
```

13.2.3 Configuring an acquisition

Once created and reserved, the acquisition object can be configured by specifying:

1. Which device registers must be acquired:
```
int dsa_acquisition_config_trace(DSA_ACQUISITION *acq, DSA_DEVICE_BASE *dev,
                                 int trace_idx, int typ, int idx, int sidx)
```

2. Which device and event act as a trigger:
```
int dsa_acquisition_config_immediate_trigger(DSA_ACQUISITION *acq,
                                             DSA_DEVICE_BASE *dev)

int dsa_acquisition_config_begin_of_movement_trigger(DSA_ACQUISITION *acq,
                                                     DSA_DEVICE_BASE *dev,
                                                     int ipol_grp, double delay)

int dsa_acquisition_config_end_of_movement_trigger(DSA_ACQUISITION *acq,
                                                   DSA_DEVICE_BASE *dev,
                                                   int ipol_grp, double delay)

int dsa_acquisition_config_position_trigger(DSA_ACQUISITION *acq,
                                            DSA_DEVICE_BASE dev,
                                            int edge, double position,
                                            int conv, double delay)

int dsa_acquisition_config_position_int64_trigger(DSA_ACQUISITION *acq,
                                                  DSA_DEVICE_BASE *dev,
                                                  int edge, eint64 inc_position,
                                                  double delay)

int dsa_acquisition_config_trace_idx_trigger(DSA_ACQUISITION *acq,
                                             DSA_DEVICE_BASE *dev,
                                             int edge, int trace_idx,
                                             double value,
                                             int conv, double delay)

int dsa_acquisition_config_trace_idx_int32_trigger(DSA_ACQUISITION *acq,
                                                   DSA_DEVICE_BASE *dev,
                                                   int edge, int trace_idx,
                                                   int value, double delay)
int dsa_acquisition_config_trace_idx_int64_trigger(DSA_ACQUISITION *acq,
                                                   DSA_DEVICE_BASE *dev,
                                                   int edge, int trace_idx,
```

```
                                                    eint64 inc_value, double delay)
int dsa_acquisition_config_trace_idx_float32_trigger(DSA_ACQUISITION *acq,
                                                     DSA_DEVICE_BASE *dev,
                                                     int edge, int trace_idx,
                                                     float inc_value,
                                                     double delay)

int dsa_acquisition_config_trace_idx_float64_trigger(DSA_ACQUISITION *acq,
                                                     DSA_DEVICE_BASE *dev,
                                                     int edge,
                                                     int trace_idx,
                                                     double inc_value,
                                                     double delay)

int dsa_acquisition_config_register_trigger(DSA_ACQUISITION *acq,
                                            DSA_DEVICE_BASE *dev,
                                            int edge, int typ, int idx, int sidx,
                                            double value, int conv, double delay)

int dsa_acquisition_config_register_int32_trigger(DSA_ACQUISITION *acq,
                                                  DSA_DEVICE_BASE *dev,
                                                  int edge, int typ, int idx,
                                                  int sidx, int inc_value,
                                                  double delay)

int dsa_acquisition_config_register_int64_trigger(DSA_ACQUISITION *acq,
                                                  DSA_DEVICE_BASE *dev,
                                                  int edge, int typ, int idx,
                                                  int sidx, eint64 inc_value,
                                                  double delay)

int dsa_acquisition_config_register_float32_trigger(DSA_ACQUISITION *acq,
                                                    DSA_DEVICE_BASE *dev,
                                                    int edge,
                                                    int typ, int idx, int sidx,
                                                    float inc_value,
                                                    double delay)

int dsa_acquisition_config_register_float64_trigger(DSA_ACQUISITION *acq,
                                                    DSA_DEVICE_BASE *dev,
                                                    int edge,int typ, int idx,
                                                    int sidx,
                                                    double inc_value,
                                                    double delay)

int dsa_acquisition_config_int32_bit_field_state_trigger(DSA_ACQUISITION *acq,
                                                         DSA_DEVICE_BASE *dev,
                                                         int typ, int idx,
                                                         int sidx,
                                                         dword low_state_mask,
                                                         dword high_state_mask,
                                                         double delay)

int dsa_acquisition_config_int64_bit_field_state_trigger(DSA_ACQUISITION *acq,
                                                         DSA_DEVICE_BASE *dev,
                                                         int typ, int idx,
                                                         int sidx,
```

```
                                                    eint64 low_state_mask,
                                                    eint64 high_state_mask,
                                                    double delay)

  int dsa_acquisition_config_int32_bit_field_change_trigger(DSA_ACQUISITION *acq,
                                                     DSA_DEVICE_BASE *dev,
                                                     int typ, int idx,
                                                     int sidx,
                                                     dword rising_edge_mask,
                                                     dword falling_edge_mask,
                                                     double delay)

int dsa_acquisition_config_int64_bit_field_change_trigger(DSA_ACQUISITION *acq,
                                                    DSA_DEVICE_BASE *dev,
                                                    int typ, int idx,
                                                    int sidx,
                                                    eint64 rising_edge_mask,
                                                    eint64 falling_edge_mask,
                                                    double delay)
```

3. How many acquisition points, which sampling frequency or acquisition duration:

```
  int dsa_acquisition_config_with_nb_points_and_total_time(DSA_ACQUISITION *acq,
                                                     int nb_points,
                                                     double total_time,
                                                     int synchro_mode)

  int dsa_acquisition_config_with_nb_points_and_sampling_time(DSA_ACQUISITION*acq,
                                                     int nb_points,
                                                     double sampling_time,
                                                     int synchro_mode)

  int dsa_acquisition_config_with_sampling_time_and_total_time(DSA_ACQUISITION *acq,
                                                     double sampling_time,
                                                     double total_time,
                                                     int synchro_mode)
```

The `synchro_mode` argument in the above functions plays an important role on the definition of the acquisition, as it defines which adjustments EDI introduces to the acquisition configuration concerning the sampling frequency and acquisition duration. It is important to note that:

- In an AccurET position controller, the sampling frequency can be 20 kHz or 2.5 kHz depending on the register;
- In the UltimET motion controller, the sampling frequency is 10 kHz.

The options available for the `synchro_mode` argument are:

- `DSA_ACQUISITION_SYNCHRO_MODE_NONE`

EDI freezes the number of acquisition points specified.

(for the EDI function `dsa_acquisition_config_with_sampling_time_and_total_time` the number of points is obtained using the formula `total_time / sampling_time`).

EDI first computes the possible sampling frequency given the type of data to acquire and device:

AccurET		UltimET	Highest sampling rate
# traces @ 20 kHz	# traces @ 2.5 kHz	# traces @ 10 kHz	
≥ 1	X	0	20 kHz
X	X	≥ 1	10 kHz
0	All	0	2.5 kHz

X: no matter

The final sampling frequency is computed to obtain a multiple of this highest value which best fits the specified sampling time. The acquisition duration is then re-computed using this frequency and the specified number of acquisition points.

- `DSA_ACQUISITION_SYNCHRO_MODE_COMMON_STI`

EDI freezes the number of acquisition points specified.

(for the EDI function `dsa_acquisition_config_with_sampling_time_and_total_time` the number of points is obtained by using the formula `total_time / sampling_time`).

EDI first computes the lowest possible sampling frequency given the type of data to acquire and device:

AccurET		UltimET	Lowest sampling rate
# traces @ 20 kHz	# traces @ 2.5 kHz	# traces @ 10 kHz	
All	0	0	20 kHz
0	0	All	10 kHz
All	0	≥ 1	10 kHz
X	≥ 1	X	2.5 kHz

X: no matter

The final sampling frequency is computed to obtain a multiple of this lowest value which best fits the specified sampling time. The acquisition duration is then re-computed using this frequency and the specified number of acquisition points.

- `DSA_ACQUISITION_SYNCHRO_MODE_MIN_STI`

EDI freezes the number of acquisition points specified.

(for the EDI function `dsa_acquisition_config_with_sampling_time_and_total_time` the number of points is obtained by using the formula `total_time / sampling_time`).

In this specific case, EDI ignores the other arguments. It simply computes the highest possible sampling frequency given the type of data to acquire and device:

AccurET		UltimET	Highest sampling rate
# traces @ 20 kHz	# traces @ 2.5 kHz	# traces @ 10 kHz	
≥ 1	X	X	20 kHz
0	X	≥ 1	10 kHz
0	All	0	2.5 kHz

X: no matter

And, the acquisition duration is then computed using this highest sampling frequency.

13.2.4 Initiating an acquisition

It is now the moment to start the acquisition. Once an acquisition has started, the device(s) specified for acquiring data will wait until the trigger conditions on the trigger device are valid before saving the data. When the trigger conditions are met, the device(s) starts to save the data. The user application should wait until the device(s) have finished saving all the data before uploading. Therefore, EDI provides two methods for initiating an acquisition:

- Synchronous method

The EDI function is blocked until the devices have finished saving the data (or a specified timeout has elapsed).

```
int dsa_acquisition_acquire_s(DSA_ACQUISITION *acq, int timeout)
```

- Asynchronous method

The EDI function returns immediately without waiting, while allowing to specify a user callback which will be called when the device(s) have finished saving the data. On the Windows operating system, it is possible to set up to 5 simultaneous asynchronous acquisitions.

```
int dsa_acquisition_acquire_a(DSA_ACQUISITION *acq, DSA_HANDLER handler,
                              void *param)
```

13.2.5 Verifying an acquisition frequency parameters computed by EDI

Before uploading the data, it might be interesting to check the actual number of acquired points, acquisition duration and sampling frequency used. Indeed, as explained before, the acquisition frequency parameters can be adjusted by EDI in the case the user specifies settings that are not actually feasible. The following functions can be used to retrieve the acquisition frequency parameters used for the acquisition.

```
int dsa_acquisition_get_real_total_time(DSA_ACQUISITION *acq,
                                        double *real_total_time)

int dsa_acquisition_get_real_nb_points(DSA_ACQUISITION *acq, int
                                       *real_nb_points)

int dsa_acquisition_get_real_sampling_time(DSA_ACQUISITION *acq,
                                           double *real_sampling_time)

dsa_acquisition_get_trace_real_nb_points(DSA_ACQUISITION *acq, DSA_DEVICE
                                         *dev,int trace_idx, int
                                         *real_nb_points)
```

13.2.6 Uploading acquired points

Once the data has been saved, the user can perform the upload in ISO or increments units.

```
int dsa_acquisition_upload_trace(DSA_ACQUISITION *acq, DSA_DEVICE *dev,
                                 int trace_idx, int table_size, double
times[],double traces[], int conv)

int dsa_acquisition_upload_int32_trace(DSA_ACQUISITION *acq, DSA_DEVICE *dev,
                                       int trace_idx, int table_size,
                                       double times[], int traces[])

int dsa_acquisition_upload_int64_trace(DSA_ACQUISITION *acq, DSA_DEVICE *dev,
                                       int trace_idx, int table_size,
                                       double times[], eint64 traces[])

int dsa_acquisition_upload_float32_trace(DSA_ACQUISITION *acq, DSA_DEVICE
                                         *dev,int trace_idx, int table_size,
                                         double times[], float traces[])

int dsa_acquisition_upload_float64_trace(DSA_ACQUISITION *acq, DSA_DEVICE
                                         *dev,int trace_idx, int table_size,
                                         double times[], double traces[])
```

13.2.7 Un-reserving an acquisition

Once the uploading is completed, the user can un-reserve the acquisition object. Although not mandatory, it is recommended to do so when several acquisition objects have been created, so that another acquisition object can be reserved, configured and used without conflicts.

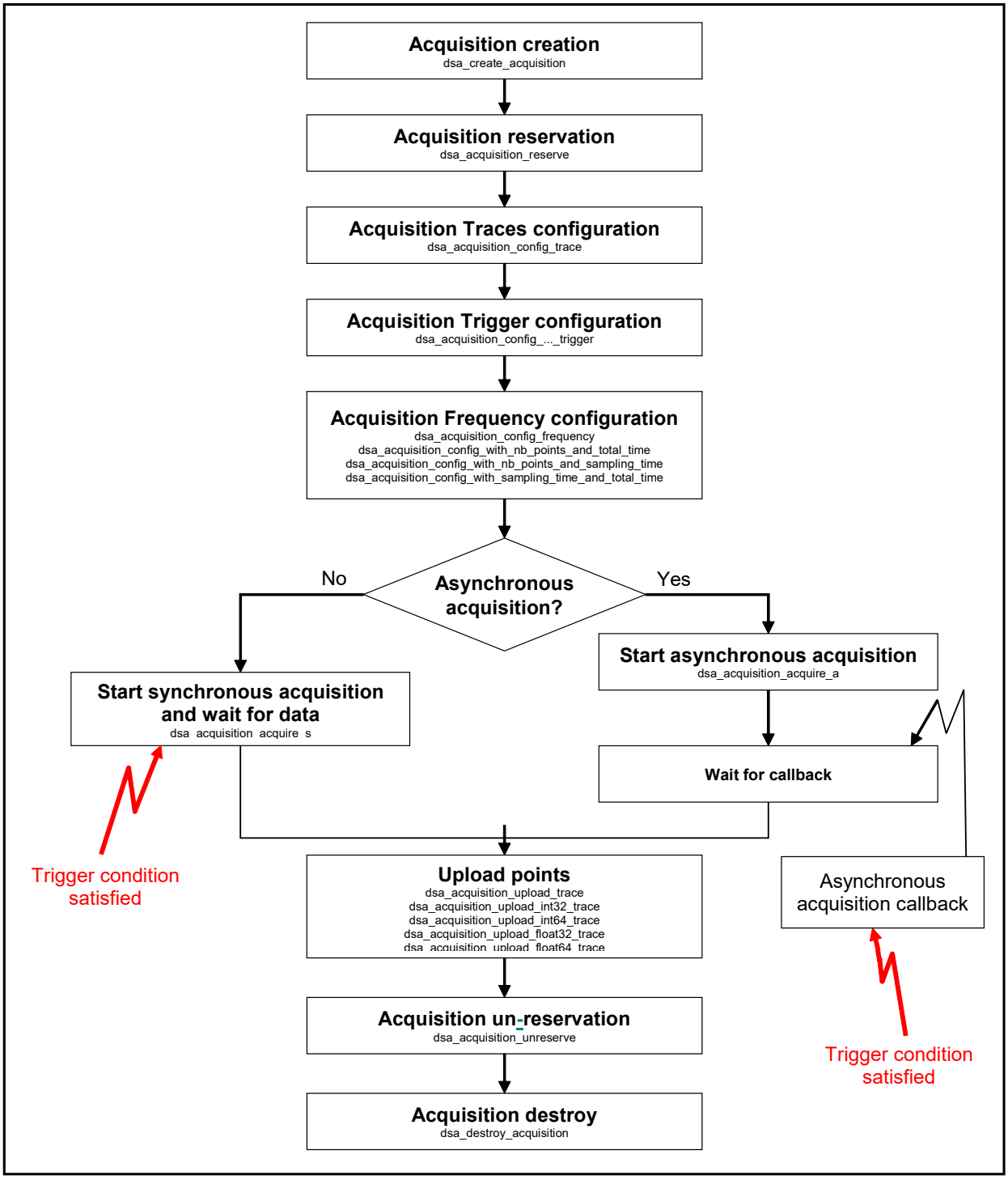
```
int dsa_acquisition_unreserve(DSA_ACQUISITION *acq)
```

13.2.8 Destroying an acquisition object

Finally, the acquisition object can be destroyed if the user application does not need it anymore. Be aware that this will free the dynamically allocated memory. If creation and destruction of acquisition objects is done continuously, it will create memory fragmentation.

```
int dsa_destroy_acquisition(DSA_ACQUISITION **acq)
```

A programming example of a normal acquisition is provided with the EDI package. Here is a flowchart depicting the normal acquisition process.



13.3 EDI programming: Continuous Acquisition

For this acquisition mode, creating and configuring traces is done using the same EDI functions as for the normal acquisition mode. The main difference is that this acquisition mode is not associated to a trigger. Therefore, the acquisition will start as soon as the start function is executed.

13.3.1 Creating an acquisition

The same object as for normal acquisition (`DSA_ACQUISITION`) can be used. Again, when creating this object, the user specifies which devices take part of the acquisition, namely:

- Devices which will acquire data;
- Devices which will synchronize the acquisition (e.g. UltimET motion controller).

The EDI function for creating an acquisition object uses dynamic memory allocation. The continuous usage of dynamic memory allocation is not recommended because of memory fragmentation and performance degradation, in particular for real-time applications. Thus, it is recommended to create the acquisition objects altogether when starting the application and destroy them just before exiting the application.

```
int dsa_create_acquisition(DSA_ACQUISITION **acq, DSA_DEVICE_BASE *dev)
```

13.3.2 Reserving an acquisition

An acquisition object must be reserved before it can be used. It is at this particular moment that EDI actually verifies if every device specified at object creation is not already taking part on another reserved acquisition. From a programming point of view, realizing this check earlier when creating the acquisition object would be too constraining considering the recommendation for creating acquisition objects altogether at application startup. With this mechanism of reservation, a user can create all the acquisition objects as recommended and reserve them in a need to use basis.

```
int dsa_acquisition_reserve(DSA_ACQUISITION *acq)
```

13.3.3 Configuring an acquisition

Once created and reserved, the acquisition object can be configured by specifying:

1. Which device registers must be acquired.

```
int dsa_acquisition_config_trace(DSA_ACQUISITION *acq, DSA_DEVICE_BASE *dev,
                                 int trace_idx, int typint idx, int sidx)
```

2. Size of the device's circular buffer and the acquisition sampling time

By default, the circular buffer's size is set to the maximum corresponding to 16384 acquisition points.

```
int dsa_acquisition_continuous_config(DSA_ACQUISITION *acq,
                                      int rolling_buffer_size,
                                      double sampling_time)
```

13.3.4 Initiating an acquisition

It is now the moment to start the continuous acquisition.

```
int dsa_acquisition_continuous_acquire(DSA_ACQUISITION *acq)
```

This function just starts the continuous acquisition and exits immediately.

13.3.5 Waiting for acquisition points

As mentioned before, as soon as the user application initiates an acquisition, the data is continuously saved to a circular buffer. On the one hand, the user application must wait until enough points have been saved before initiating the upload. On the other hand, the user application must upload the traces fast enough to avoid overrunning already acquired samples (data loss).

There are two methods available for waiting for available points:

- **Polling method**

The user polls the device for the number of points already acquired and when sufficient data is available, starts the uploading. The following function returns the number of available points:

```
int dsa_acquisition_continuous_get_nb_available_values(DSA_ACQUISITION *acq,
                                                       int *nb_data)
```

- **Waiting method**

EDI provides functions for waiting until the specified number of points has been stored in the circular buffer:

```
int dsa_acquisition_continuous_wait_available_values_s(DSA_ACQUISITION *acq,
                                                       int nb_available_values,
                                                       long timeout)

int dsa_acquisition_continuous_wait_available_values_a(DSA_ACQUISITION *acq,
                                                       int nb_available_values,
                                                       DSA_HANDLER handler,
                                                       void *param)
```

13.3.6 Uploading acquired points

Once data is available, the user application can upload the points using one of the following functions. Note that it is mandatory to upload the same number of points as it was specified in the waiting functions described before. Furthermore, the size of the arrays provided to the must sufficient to store the uploaded points.

```
int dsa_acquisition_continuous_upload_int32_trace(DSA_ACQUISITION *acq,
                                                  DSA_DEVICE *dev, int trace_idx,
                                                  Int nb_available_values,
                                                  double times[], int traces[])

int dsa_acquisition_continuous_upload_int64_trace(DSA_ACQUISITION *acq,
                                                  DSA_DEVICE *dev, int trace_idx,
                                                  int nb_available_values,
                                                  double times[], eint64
traces[])

int dsa_acquisition_continuous_upload_float32_trace(DSA_ACQUISITION *acq,
                                                    DSA_DEVICE *dev, int trace_idx,
                                                    int nb_available_values,
                                                    double times[],float traces[])

int dsa_acquisition_continuous_upload_float64_trace(DSA_ACQUISITION *acq,
                                                    DSA_DEVICE *dev,
                                                    int trace_idx,
                                                    int nb_available_values,
                                                    double times[],
                                                    double traces[])
```

```
int dsa_acquisition_continuous_upload_trace(DSA_ACQUISITION *acq, DSA_DEVICE *dev,
                                            int trace_idx,
                                            int nb_available_values,
                                            double times[], double traces[],
                                            int conv)
```

13.3.7 Checking integrity of the uploaded data

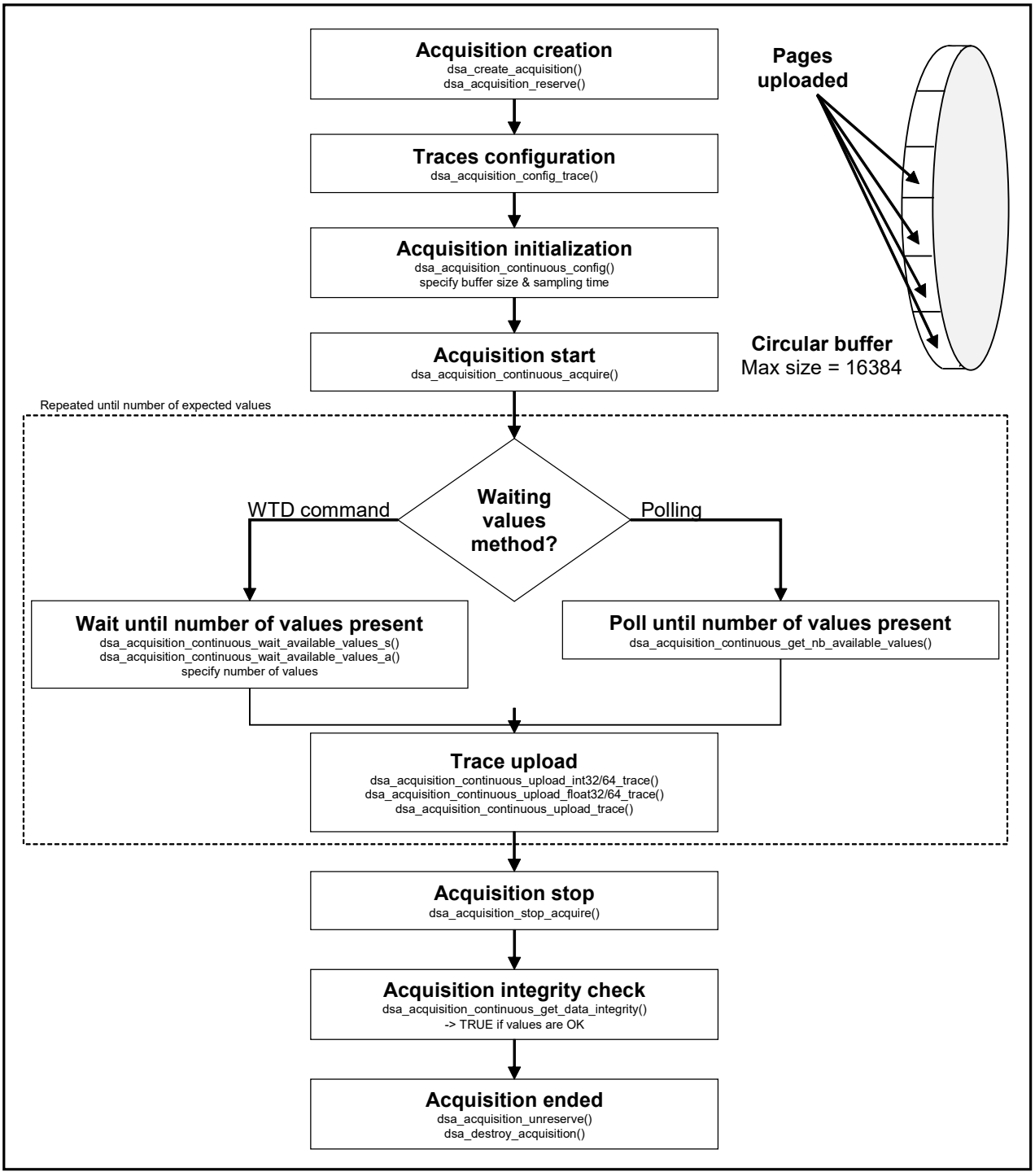
If the user application is not sufficiently fast to upload the traces from the controller, it could occur that newer samples overwrite older samples in the circular buffer, corresponding effectively to data loss. The user can verify the data integrity with the following function:

```
int dsa_acquisition_continuous_get_data_integrity(DSA_ACQUISITION *acq,
                                                  ebool *integrity_guaranteed)
```

The next section will provide guidelines for configuring correctly the continuous acquisition to avoid data loss (samples overrun on the circular buffer).

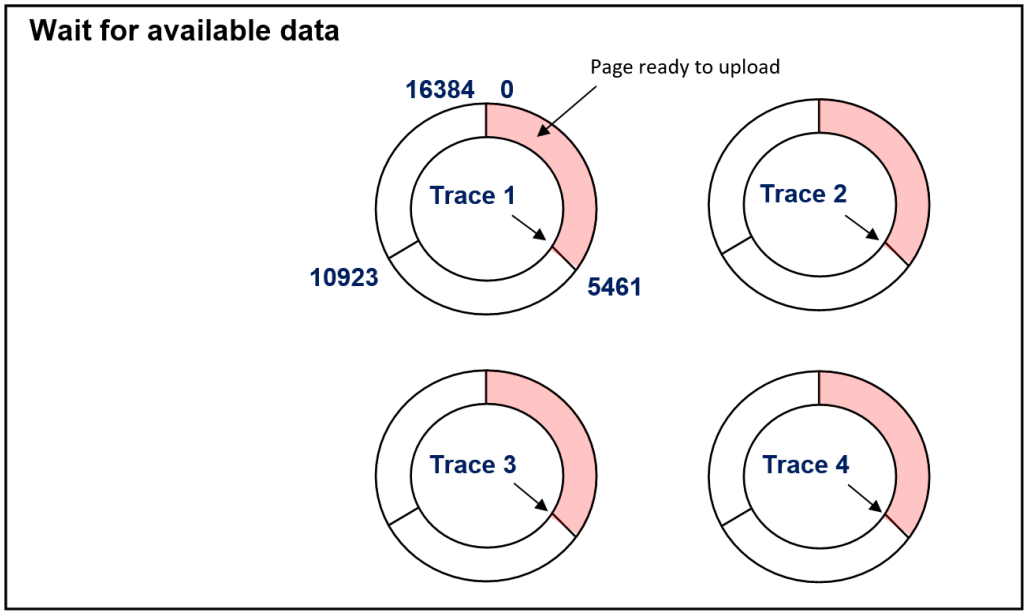
13.3.8 Guidelines for configuring a continuous acquisition

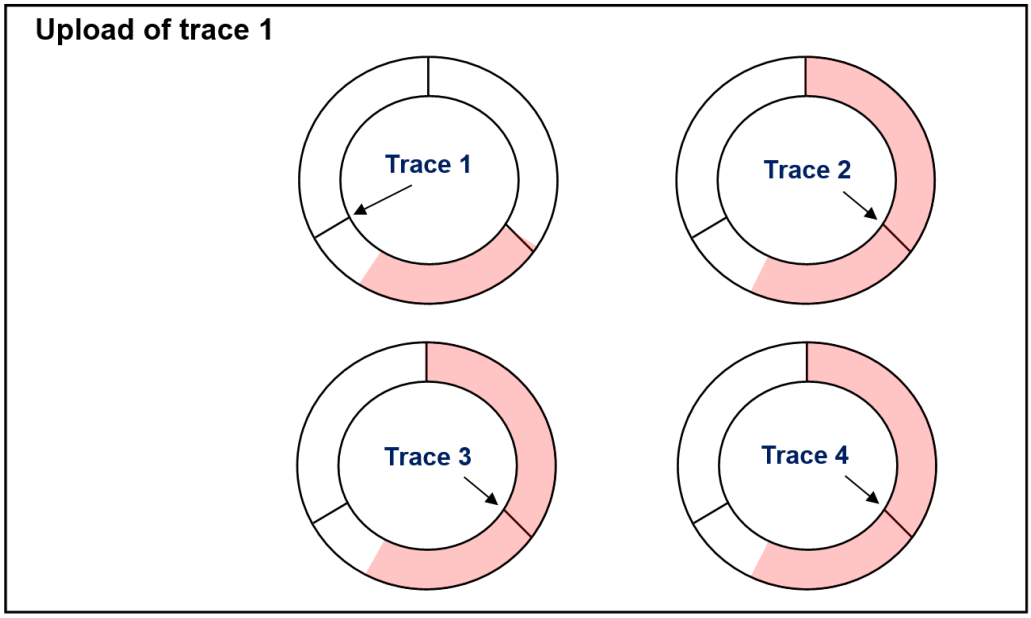
The flowchart below depicts the continuous acquisition process:

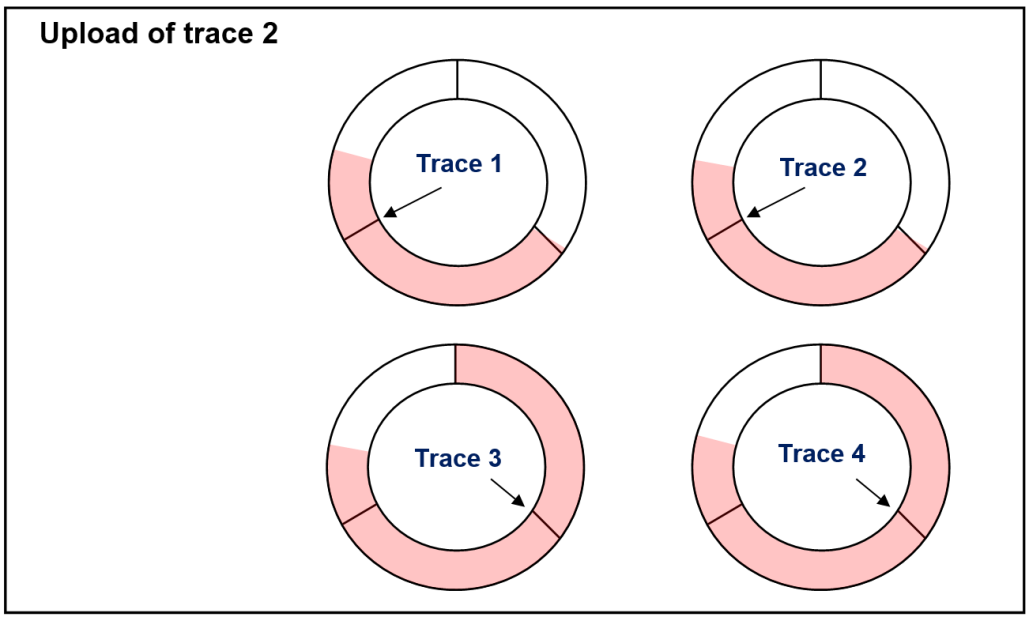


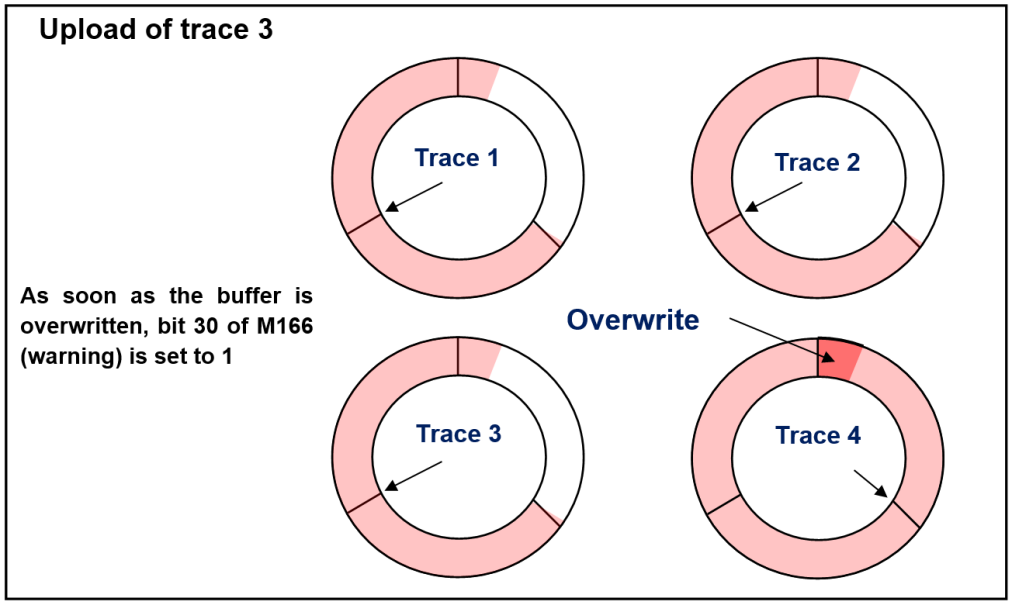
The critical aspect of a continuous acquisition is to find the correct configuration to avoid the overwriting of data in the circular buffer.

Example of an upload not sufficiently fast:
- 4 traces (i.e. 4 circular buffers).
- Maximum buffer size (= 16384 points).
- Waiting for 1/3 of the buffer's size for available data.


Wait for available data
Page ready to upload
16384
0
10923
5461
Trace 1
Trace 2
Trace 3
Trace 4


Upload of trace 1
Trace 1
Trace 2
Trace 3
Trace 4


Upload of trace 2
Trace 1
Trace 2
Trace 3
Trace 4



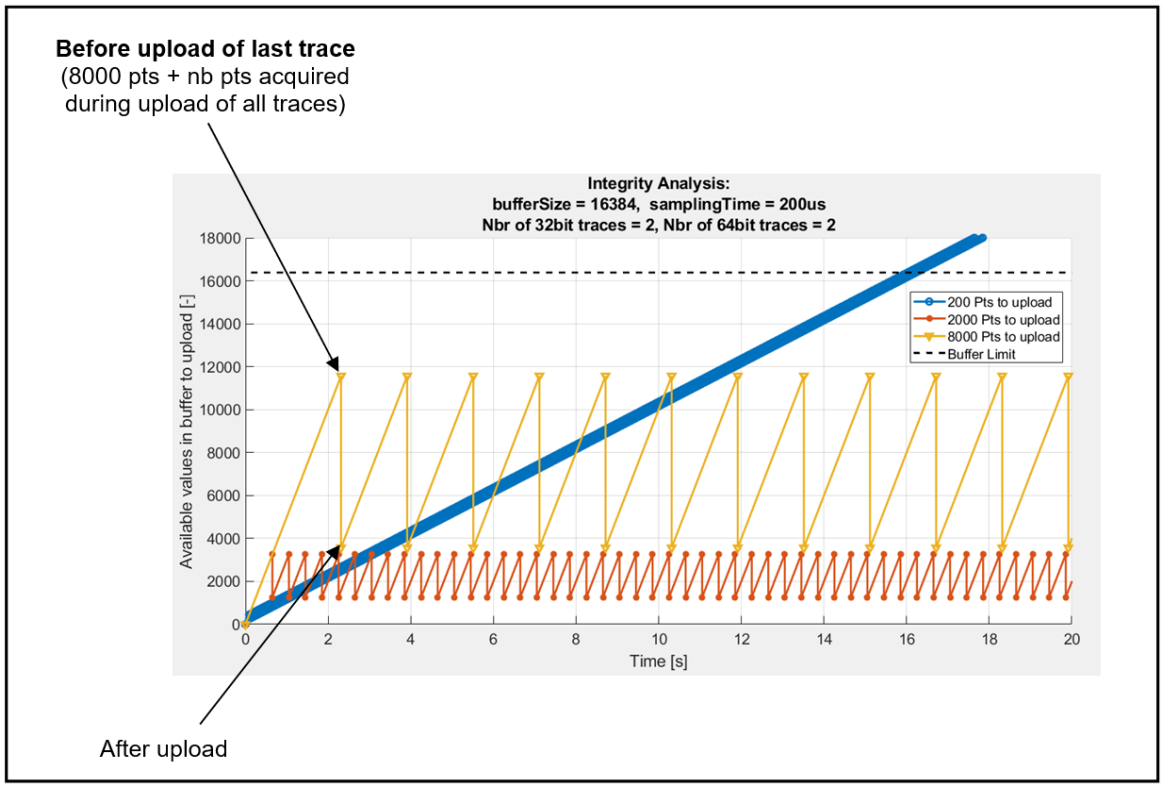
13.3.8.1 Theoretical analysis

The following factors impact the behavior of the continuous acquisition:

1. Hardware and software configuration (PC and Operating System performance, in particular for the upload time and the storage media used for storing the uploaded data).
2. Number of traces.
3. Buffer size (usually it is set to its maximum of 16384 points).
4. Sampling time.
5. Number of points to wait for before uploading (NBPU).

Usually, the requirements of the application determine the three first factors. The only factors the user can tweak are the **sampling time** and **NBPU**. Therefore, the goal of the tuning is to determine the best values for these settings.

The following graph shows the theoretical behavior of a system with a fixed sampling time and different NPBU.



13.3.8.2 Tuning guidelines

Follow these guidelines to configure a continuous acquisition:

1. Define the Hardware/Software configuration (PC type, Operating System…).
2. Define the device(s) registers which must be acquired.
3. Define if a continuous acquisition is really necessary or if it can be replaced by a "long" normal acquisition.
4. Define the maximum allowed acquisition sampling time. The longer the sampling time, more time will the application have to upload the traces.
5. Measure the time the application needs to upload and save all traces of n points ((NBPU). Build a graph for different values of n.
6. Define a sampling time and a NPBU which best respects the following rule:
   Time to upload and save all traces of NBPU points each < sampling time * NBPU
7. Validate the configuration with the final real using EDI continuous acquisition functions.

Remark: If following these guidelines it is not possible to find a configuration for the continuous acquisition, the requirements of the application must be revised (refer to points 1. and 2. of the above listing).

14 Real-time applications

Some EDI-based applications need to comply with strict timing requirements. For addressing such use cases, EDI includes support for the RTX real-time operating system. Refer to §2.2 for information about the supported operating systems.
To efficiently execute an EDI-based program on RTX it is necessary to setup a specific RTX subsystem. The next section explains how to configure RTX for obtaining the best real-time performances.

However, it is wrong to assume that running such programs on a real-time operating system is for itself sufficient for achieving the level of performance required by certain applications, particularly if the timing requirements are extremely tight (measured in the microsecond scale). Indeed, in such cases the architecture of the software plays an equally important role and cannot be disregarded. This chapter provides a few useful guidelines on how to design a real-time program based on EDI.

Finally, this chapter concludes with a programming example of an EDI-based real-time application.

Remark: The PCIe interface of the ULTIMET ADVANCED motion controller is not supported on the RTX operating system.

14.1 RTX subsystem configuration

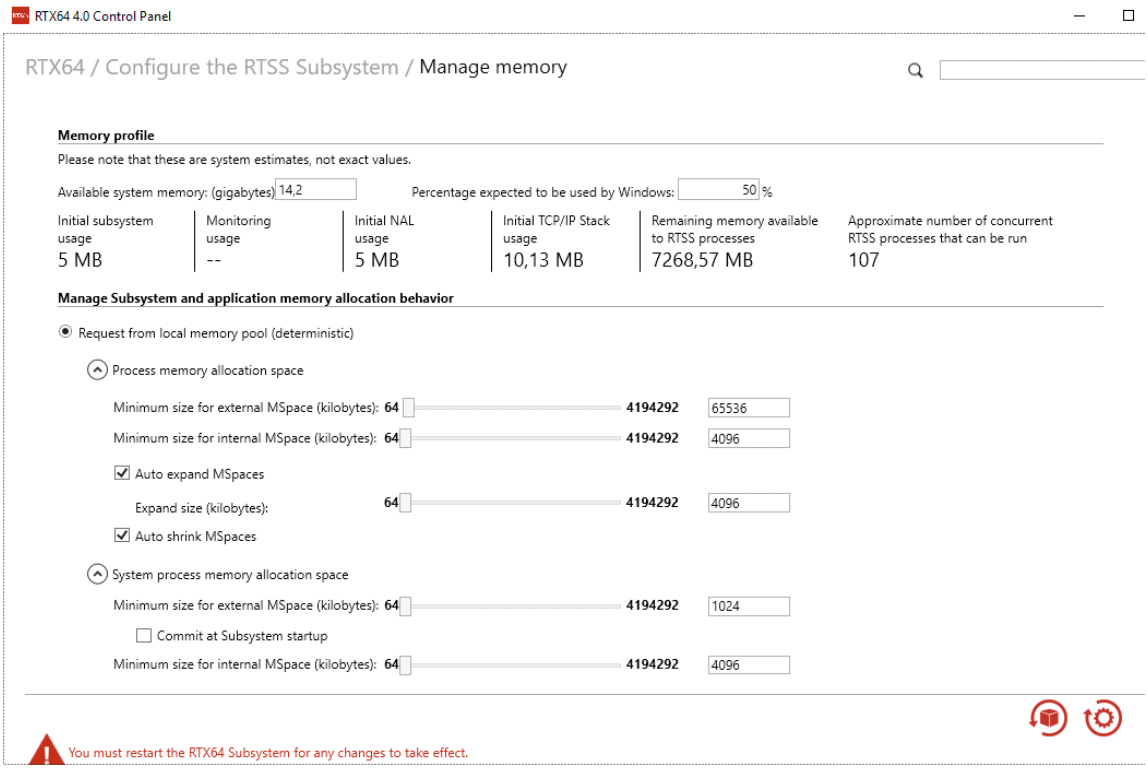
Except for the changes described next, the RTX subsystem default configuration can be used.

14.1.1 Memory Allocation Behavior

It is recommended to configure carefully the memory allocation behavior. Check RTX documentation about memory configuration. A balance must be found between the number of RTSS processes and the available memory. When using EDI, at least 256MB should be reserved for external MSSpace.

- Select the option **Request from local memory pool** (deterministic);
- Set the **Local memory pool size (kilobytes)** to 65536;
- Select the option Auto expand and set the Expand size (kilobytes) to 1024;
- Select the option Auto Shrink.

The figure below depicts such settings.

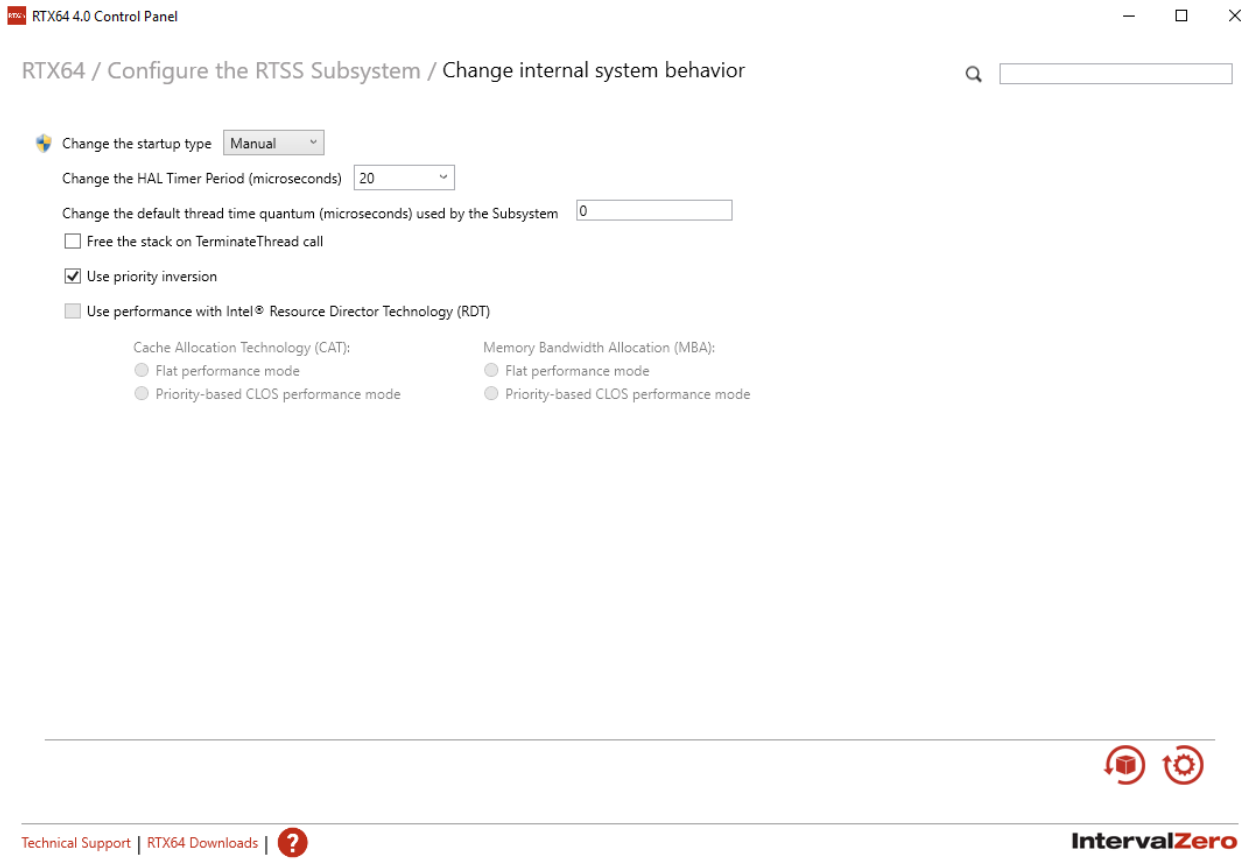


With this setting, the dynamic allocation of memory uses RTX's internal memory pool which is requisitioned from Windows at startup. This renders the creation of EDI objects much faster.

14.1.2 Internal System Behavior

It is recommended to configure the internal system behavior as follows:

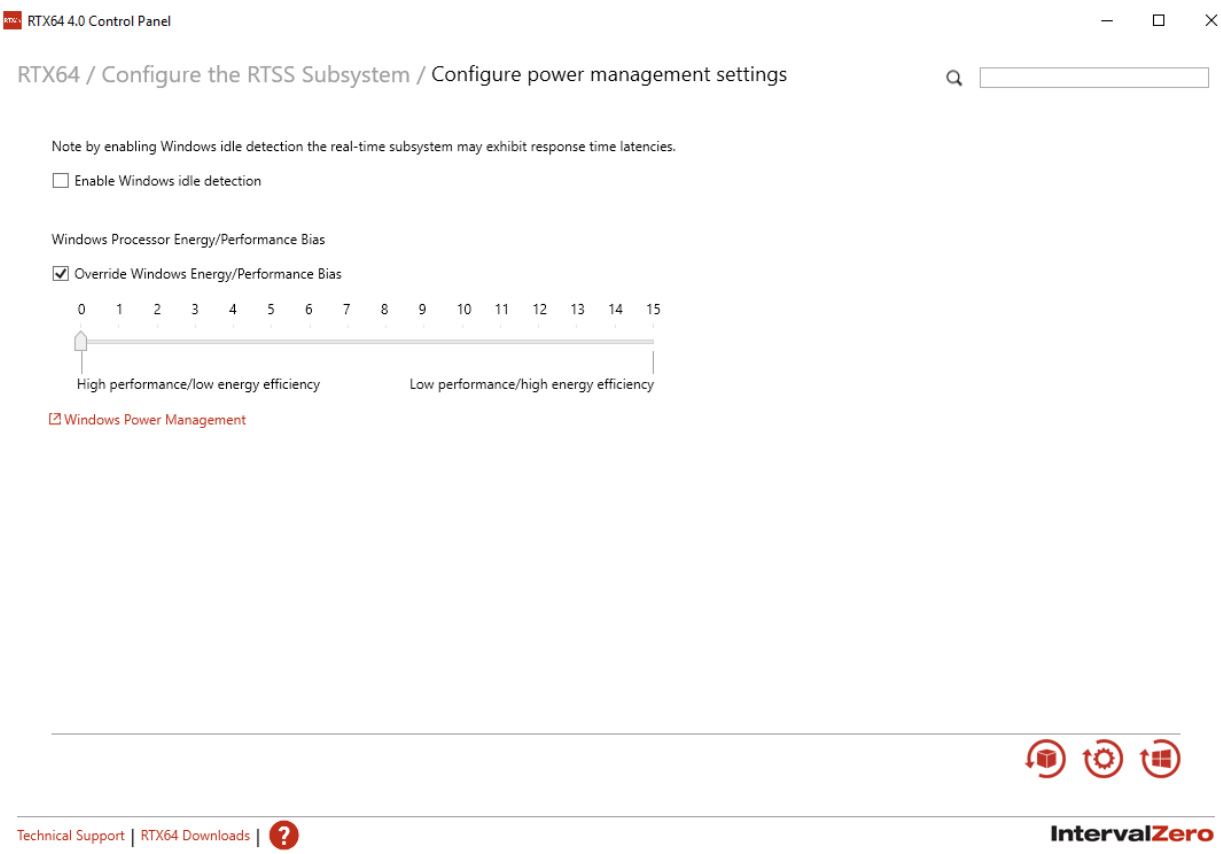
- Select the **Manual** option from the **Change the startup type** drop-down list;
- **Change the HAL timer Period (microseconds)** to 20 (IntervalZero recommends this value for applications that need to be very responsive to interrupts);
- **Change the default thread quantum (microseconds) used by the Subsystem** to 0 (zero);
- Unselect the option **Free the stack on TerminateThread call**;
- Select the option **Use priority inversion**.



14.1.3 Power management

It is recommended to configure the power management as follows:

- Select the Override Windows Energy/Performance Bias;
- Place the slide completely to the left (High performance/low energy efficiency)



14.2 RTX application

Before designing a real-time application, it is important to identify the different tasks it should realize and categorize these according to the following notions.

14.2.1 Non-real time

When the reaction time of the application (or task) to events is not critical and does not impact the proper execution of the process for which the machine is intended.

14.2.2 Real-time

An application (or task) that works in real-time can still afford execution latencies and non-determinism without jeopardizing system operation, eventually just degrading it marginally. The latencies experienced are usually in the range of a few hundreds of microsecond up to a few milliseconds.

14.2.3 Hard real-time

A hard real-time application (or task) must really comply with the timing execution requirements otherwise the system fails to operate. Tasks are typically executed cyclically at 10 kHz, with acceptable latencies up to a few tenths of microsecond.

A classic example of a hard real-time application is one where the trajectory generation is not done at the Controller level; instead it is the host PC that computes the trajectory setpoints for performing application specific motion profiles. The successive position setpoints composing the trajectory are then transmitted to the Controller(s) in a timely manner (typically one setpoint every 100 µs) to achieve the desired dynamic motion. Missing a setpoint or delaying it too much degrades the trajectory accuracy, possibly even triggering a tracking error on the Controller(s).

14.2.3.1 Cyclic interrupt user callback function

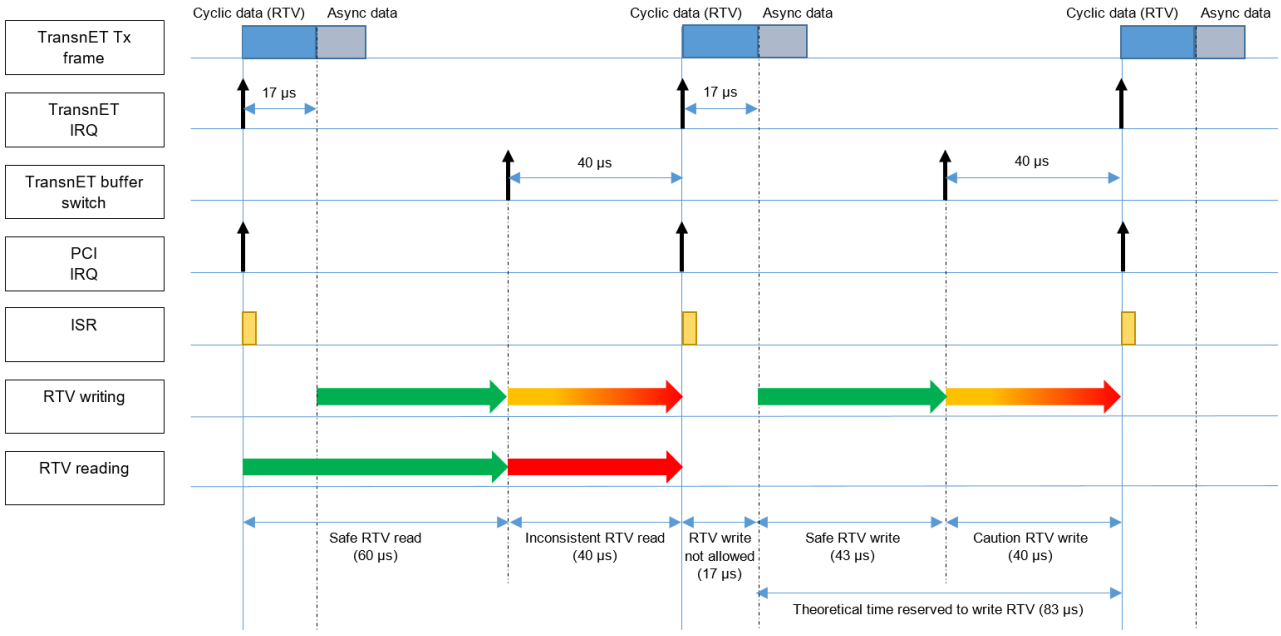
To realize the hard real-time application described above configure an UltimET motion controller to generate a PCI/PCIe cyclic interrupt (IRQ) synchronized with the TransnET cycle (10 kHz) and attach a user callback function (refer to §14.4 for the programming details). In the user callback function, send the previously computed trajectory setpoints to the Controller(s) using the real-time values (RTV) mechanism (refer to the "UltimET Light User Manual" for information about RTVs). Finally, the Controller(s) will drive the axis to the target setpoint using its internal position control loop.

The following timing diagram depicts the execution of the different elements involved on this transaction. The TransnET cycle represents the time reference. The UltimET motion controller generates a PCI/PCIe IRQ synchronized with the TransnET cycle. The RTX operating system reacts to this IRQ executing the ISR routine, which in turn triggers the IST task (refer to §14.3.3 for further information about these tasks). The IST task analyzes the root cause of the IRQ and executes the user callback function in the case of a cyclic IRQ.

Remark: The PCI/PCIe IRQ generation can be offset with respect to the TransnET IRQ, by chunks of 30 ns (refer to §14.3.4 and §14.4.3.1 for further information).

To ensure optimal real-time execution, the host PC must respect certain time constraints when writing/reading RTVs:

- RTVs must be written after the cyclic data of the current TransnET frame is dispatched (≥17 µs) and before the start of the next TransnET cycle.
- RTVs must be read before the TransnET buffer switch is executed (≤ 60 µs) to ensure that the values received are indeed valid.



TransnET IRQ, Operating System and user callback timing (theory)

14.2.3.2 Execution latencies

It is important to understand that in a real system there are always latencies and some non-determinism on the execution of a program, even when using a real-time operating system like RTX. It is not possible to provide an exact value to quantify these latencies because it depends on many different factors like for e.g., the hardware being used (chipset, processor, how many and which devices are plugged, which interfaces are used, etc.) and the software (how many tasks are running, their priorities, is there a lot of context switch, how the code is optimized, etc.). But, as an order of magnitude, it can be said that it typically varies from a few to some tenths of microsecond.

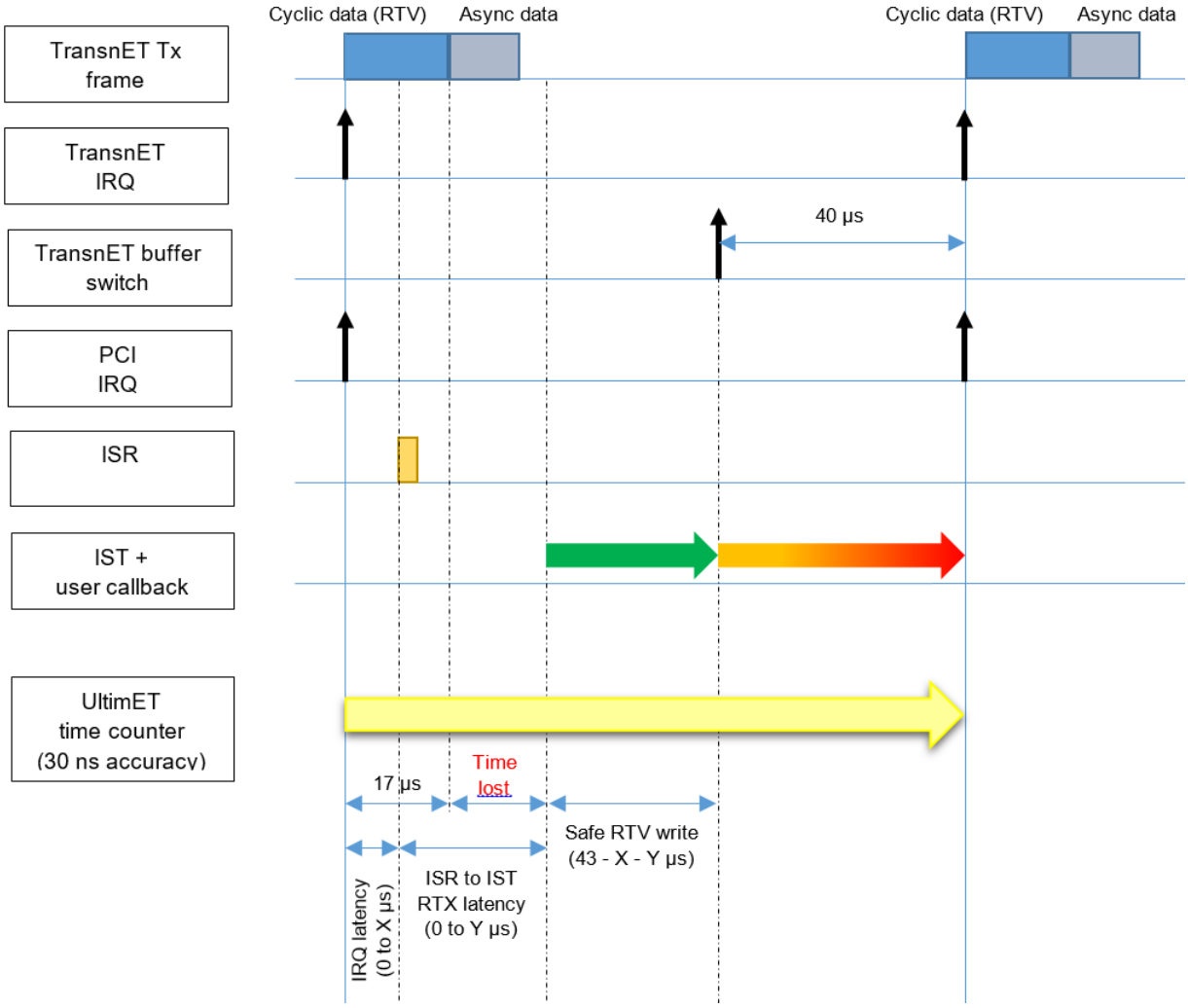
Of course, for hard real-time applications, it is important to take these additional delays into account, as to better optimize the application to comply with the timings constraints presented above for writing and reading RTVs.

One of the following methods can be used:

- **Polling**
  UltimET PCI/PCIe includes a precise time counter which keeps track of the time elapsed (in chunks of 30 ns) since the PCI/PCIe IRQ was generated and it is reset at every new IRQ. This counter can be polled in a loop until it reaches 17 µs. From this point on, RTVs can be safely written. If the latencies accumulated up to the execution of the user callback function are greater than 17 µs, the loop is not executed and the RTVs are written immediately. On the other hand, if these latencies are less than 17 µs, this approach is not very efficient since the loop execution is blocking the CPU, preventing it from executing other tasks.

- **Offset the PCI/PCIe interrupt generation with respect to TransnET**
  The alternative to the inefficient polling is to directly offset the PCI/PCIe IRQ generation with respect to TransnET IRQ by just the right amount to ensure that when the user callback function is executed, the RTVs can be written immediately. So, understanding the latencies experienced by the system is important to determine the "optimal and safe" offset to set.

The next timing diagram depicts the latencies inherent to the system. It is clear the effect of these latencies on "narrowing" the time interval available for safely writing the RTVs.



TransnET IRQ, Operating System and user callback timing in reality

14.3 Application design guidelines

14.3.1 Dynamic memory allocation

EDI functions can be grouped into 3 categories:

- Functions realizing dynamic memory allocation.
- Functions realizing dynamic memory deallocation.
- All other functions.

An application should be designed to avoid repetitive dynamic memory allocation/deallocation, as this will lead to memory fragmentation and, eventually, failure. Please refer to §3.6 for further information.

Remark: To avoid memory fragmentation it is recommended to allocate all the memory required only once at program startup and deallocate when no longer required just before program exit. Refer to EDI's HTML documentation to determine if a specific EDI function realizes dynamic memory allocation/deallocation.

14.3.2 Communication with the Controllers

As a rule of thumb, optimize the communication with the Controllers. This can be accomplished in several ways. For example, if different parts of the application need to frequently monitor various Controllers' registers, the best is to create a single task which polls regularly for the required registers and stores them in a memory pool where the other tasks can access. Another example is the use of the RTV functionality (refer to §14.4.3.1 or "UltimET Light User Manual" for further information).

Furthermore, it is highly recommended to reduce the communication with the Controllers to the strict minimum during the phases of execution where the hard real-time capability must be ensured. Therefore, the following actions should be avoided:

- Sending commands.
- Reading/Writing registers (use the RTV mechanism instead).
- Triggering Status Change. Although, it is not possible to stop all Status Change for occurring (for e.g. Status Change due to errors, warnings, as well as moving or in position status bit changes cannot be avoided), their occurrence can be nevertheless reduced. Keep in mind that running Sequences can also trigger Status Change and User Status Change should be avoided as well.

14.3.3 Task priority

The real-time sections of an application should be assigned to a task with higher priority. Adjusting the priority of tasks should be done with extreme caution: a task with high priority can starve the operating system if it never waits for the availability of resources.

EDI (RTX) creates several internal tasks:

- **IST**: this task is called by the RTX operating system at each PCI/PCIe IRQ (via the ISR routine). EDI sets the priority level for this task to the highest possible (127). The user cannot modify it. This task is also responsible for calling the user callback function (if one has been configured). Please refer to §14.4.3.1 for further information.
- **ETB**: this task manages the communication with the Controllers. More specifically, it handles *Command Acknowledgements and Status Changes*. By default, the priority level is set to 126. The user can modify it, but it is not recommended (reducing the priority level degrades the communication performance with the Controllers).
- **DSA asynchronous command management**: this task handles the *Command Acknowledgements* of all asynchronous commands (*dsa_..._a* functions). By default, the priority level is set to 108. The user can modify it, but it is not recommended (reducing the priority level degrades the performance of the asynchronous communication with the Controllers).
- **DSA asynchronous acquisition management**: EDI creates five additional tasks for handling asynchronous acquisitions. The priority level of these tasks is set to 100. The user cannot modify it.

Remark: The user can create additional tasks required by the application. But, it is important to consider the priority level set for these tasks. Tasks with a very high priority (≥ 100) can impact negatively the behavior of EDI.

14.3.4 Tuning the User callback

As mentioned before, the UltimET motion controller can be configured to delay the PCI/PCIe IRQ generation with respect to the TransnET cycle. With this feature, the execution of the user callback function can be tuned to start at the right moment to optimize the writing of RTVs.

Follow this procedure to optimize the application:

- Find the minimum time at which the user callback function is executed with respect to the PCI/PCIe IRQ:

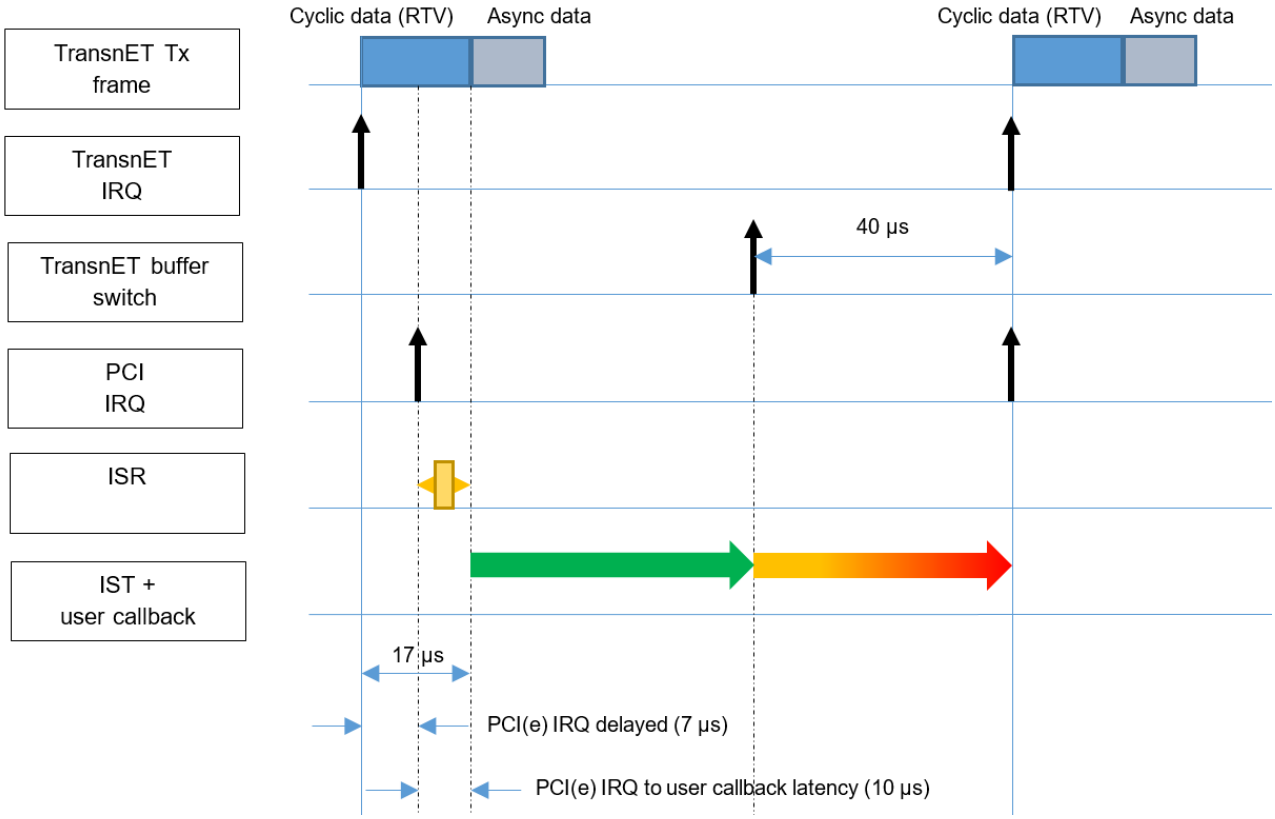
- Modify the user callback function to read the value of the UltimET's time counter when entering the function.
- Configure the PCI/PCIe IRQ with a delay of 0 µs with respect to the TransnET cycle and run the application for several minutes. When running always keep track of the absolute lowest counter value with the variable `minCallbackStart`.

- If `minCallbackStart ≤ 17 µs`, configure PCI/PCIe IRQ generation delay for the value `delay = 17 µs – minCallbackStart`.
- In the user callback function, before writing the RTV, poll in a loop the UltimET's time counter until it reaches `delay`.

In fact, the polling loop should never wait because the user callback actually starts at:

```
User callback start time = delay + minCallbackStart
                         = 17 µs – minCallbackStart + minCallbackStart
                         = 17 µs
```

The following timing diagram depicts the example where `minCallbackStart` is measured at 10 µs, so the PCI/PCIe IRQ generation offset is set to 7 µs (= 17 µs – 10 µs).



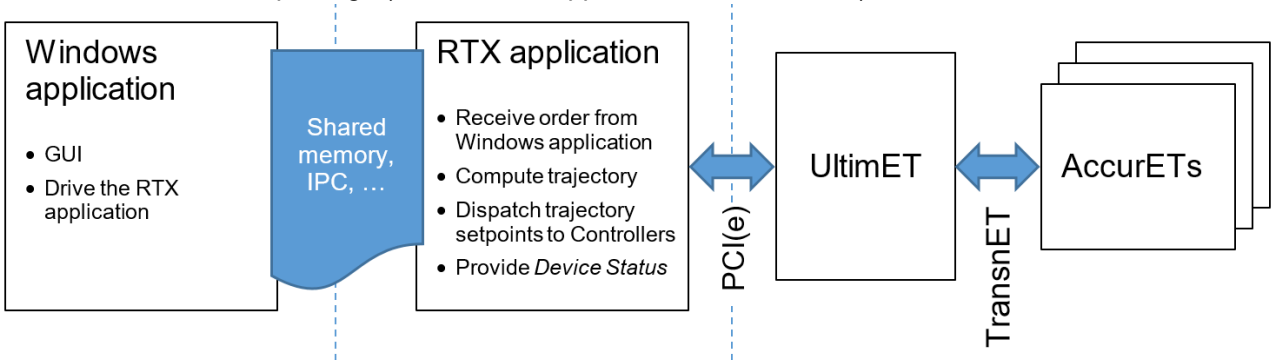
Remark: If it is also intended to read RTVs **before** writing RTVs, this procedure is not suited. Instead, remove the delay for generating the PCI/PCIe IRQ and first read the RTVs when executing the user callback function. Then, do a polling loop of the UltimET's time counter to check if 17 µs have elapsed before writing the RTVs.

14.3.5 Synchronization objects

The user callback function should execute as fast as possible and never overflow to the next IRQ cycle. Therefore, the use of synchronization objects like semaphore or mutexes is highly disapproved. The risk is that a low priority task owns the synchronization object, blocking the execution of the user callback function.

14.4 Example

This section includes an example of a RTX real-time application based on EDI. The source code is provided with the EDI distribution package (the Windows application is not included).



14.4.1 Specification

This application must create a circle trajectory and store the [X, Y] positions in a 2D array in a shared memory. The trajectory setpoints must be dispatched to the Controller in real-time at a rate of 10 kHz. The Controller must drive two motors, one for each axis.

In return, the application must receive the Controller status and position of the axes.

Remark: In a real implementation, the trajectory setpoints would normally be pre-computed by the Windows application and shared with the RTX application. The RTX application would then dispatch the setpoints to the Controllers in real-time. Similarly, the status and position information would be displayed in the Windows application GUI. In this example, and for the sake of simplificity, both of these tasks are realized by the RTX application.

14.4.2 Design

As explained before when designing a real-time application, it is important to identify the different tasks it should realize and categorize these according to:
- Non real-time;
- Real-time; and
- Hard real-time.

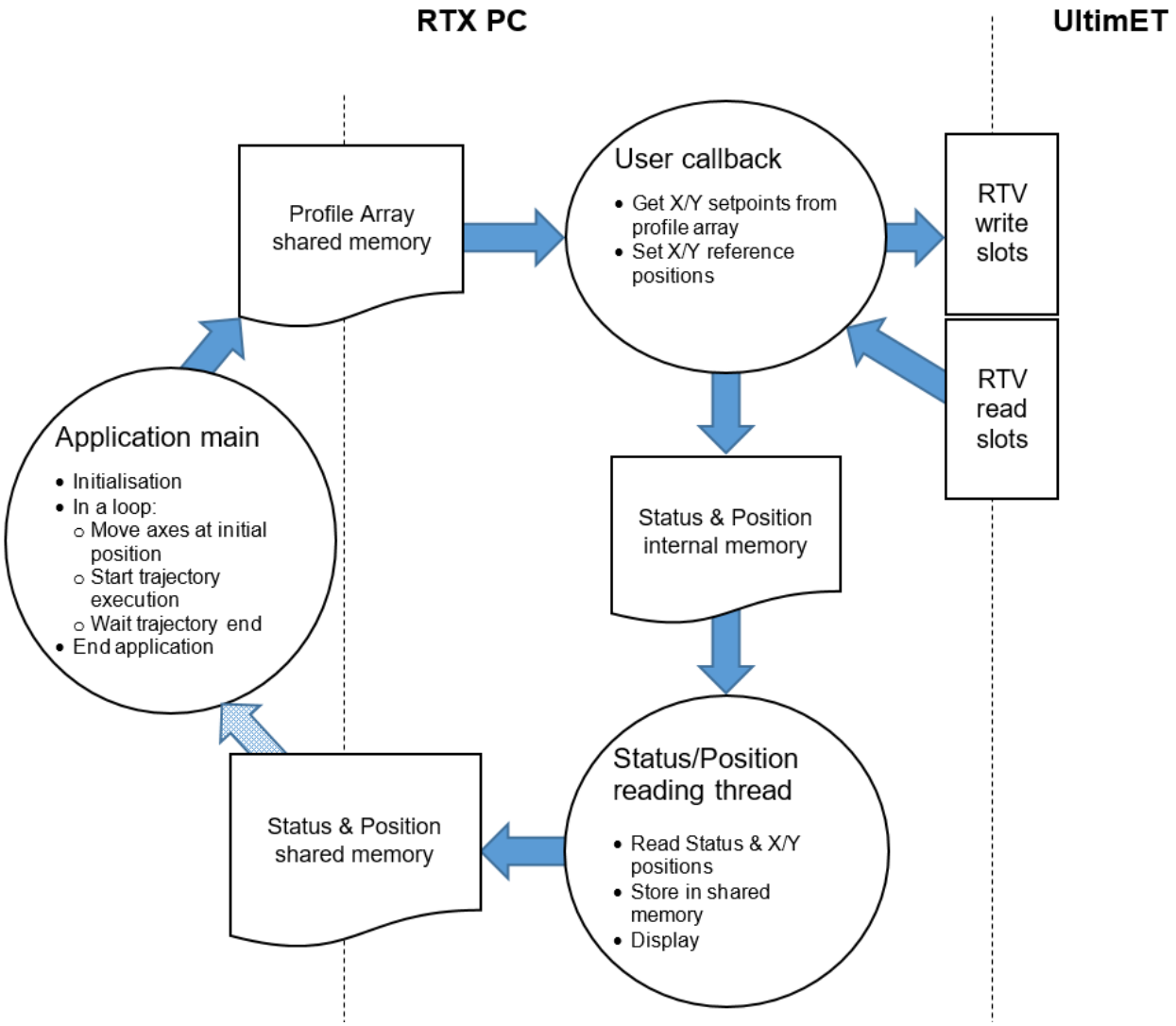
For this example, the non real-time tasks are mostly linked to the initialization of the application (memory allocation, Controllers reconfiguration, axes power up, etc.) and cleaning up before exiting (axes power down, memory deallocation, etc.).

The reception of Controller status and position axes are considered real-time tasks. The information must be handled regularly, but the application can cope with longer and less deterministic delays without jeopardizing system operation.

Finally, the hard real-time portion comprises the dispatching of the trajectory setpoints to the Controller. This information has to be provided regularly and deterministically to the Controller at the rate of 10 kHz.

In a nutshell, this example explains how to:
- Set an AccurET controller in a mode to accept external position references.
- Create (and destroy) real-time objects and how to read and write them.
- Setup a cyclic IRQ and user callback function at the required frequency.
- Write and read RTVs at the right moment.

The diagram below gives an overview of the implementation.



14.4.3 Programming

This application is spread over 3 modules:
- Real-time management.
- Status management.
- Application main.

14.4.3.1 Real-time management

This module includes all functions related to the real-time execution:
- Creation of the real-time objects.
- Computation of the trajectory setpoints.
- User callback.
- Enabling/Disabling the execution of the real-time trajectory.

Creation of real-time objects

EDI provides two levels to interface real-time objects:

DSA High level interface: DSA_RTV_DATA

With this set of functions, creating an object reserves 1 or 2 slots from the master (UltimET) and links the slots to the specified device register. Specifying 64-bit registers will automatically reserve 2x 32-bit slots from the master. The register to which the object is linked defines then the type of the object: integer 64-bit, integer 32- bit, float 64-bit, float 32-bit.

It is possible to create a `DSA_RTV_DATA` object for reading. In this case the device register value is provided in the real-time TransnET slot. Reading the object reads the value from the real-time slot.

It is possible to create a `DSA_RTV_DATA` object for writing. In this case writing the object writes the value in the real-time TransnET slot which is assigned to device register.

DSA Low level interface: DSA_RTV_SLOT

With this set of functions, creating an object reserves slot(s) from the master (UltimET). This interface provides functions to create 32-bit or 64-bit slots.

The object can be manually linked to a device register of the corresponding type. Then, reading or writing `DSA_RTV_SLOT` objects provide the same functionalities as `DSA_RTV_DATA` objects.

Both interfaces provide functions to read an array of objects. The only restriction is that the array must contain objects of same type.

An ETB low-level function allows to read or write an array of 32-bit slots. This function can be used when there a mix of 64- bit and 32-bit real-time objects or a mix of integer and floating real-time objects. This allow to read or write all slots in a single call, which is much more efficient than reading/writing the different RTV types sequentially.

A drawback of this function is that one must know the slot numbers that are used and build the 32-bit slot values. In the case of 64-bit values, one must also know which 32-bit slot number corresponds to most significant DWORD and which corresponds to less significant DWORD in order to reconstruct the value.

This application example will use:

- DSA low level functions to initialize the RTVs containing the position setpoint of the X (`XReference`) and Y (`YReference`) axes. The AccurET controller will use these as reference positions.
- DSA high level functions to initialize the position feedback and status RTVs (`XFeedback`, `YFeedback`, `XStatus` and `YStatus`). These RTVs are linked to register values.
- ETB low-level function to read and write RTVs in the user callback function because it is more efficient. For this reason it is necessary to retrieve the raw slot numbers from the created objects.

Creating `XReference` and `YReference` real-time objects (EDI low-level interface):

```
//Create a slot into which the user callback write the reference position on X
if ((err = dsa_get_64bit_rtv0_slot(devices->master, &xReference))) {
   DSA_EXT_DIAG(err, devices->master);
   goto _error;
}
//Retrieve the low layer slot numbers to be able to use low-level ETB write
//function
if ((err = dsa_get_64bit_rtv_slot_nr(xReference,
                                     &writeSlotNbs[X_REFERENCE_LSL_SLOT_IDX],
                                     &writeSlotNbs[X_REFERENCE_MSL_SLOT_IDX]))) {
   DSA_EXT_DIAG(err, devices->master);
```

```
    goto _error;
}
//Create a slot into which the user callback write the reference position on Y
if ((err = dsa_get_64bit_rtv0_slot(devices->master, &yReference))) {
    DSA_EXT_DIAG(err, devices->master);
    goto _error;
}
//Retrieve the low layer slot numbers to be able to use low-level ETB write
//function
if ((err = dsa_get_64bit_rtv_slot_nr(yReference,
                                     &writeSlotNbs[Y_REFERENCE_LSL_SLOT_IDX],
                                     &writeSlotNbs[Y_REFERENCE_MSL_SLOT_IDX]))) {
    DSA_EXT_DIAG(err, devices->master);
    goto _error;
}
```

Creating `XFeedback`, `YFeedback`, `XStatus` and `YStatus` real-time objects (EDI high-level interface):

```
//Create a slot and link it to ML0 (position feedback) of axis X
//User callback reads this slot and put the read value in internal memory
if ((err = dsa_create_rtv_read(devices->x, DMD_TYP_MONITOR_INT64, 0, 0, &xFeedback))) {
    DSA_EXT_DIAG(err, devices->x);
    goto _error;
}
//Retrieve the underlaying slot object to be able to retrive the low layer slot number
if ((err = dsa_get_rtv_slot_of_rtv(xFeedback, &helpSlot))) {
    DSA_EXT_DIAG(err, devices->x);
    goto _error;
}
//Retrieve the low layer slot numbers to be able to use low-level ETB write function
if ((err = dsa_get_64bit_rtv_slot_nr(helpSlot, &readSlotNbs[X_FEEDBACK_LSL_SLOT_IDX],
                                     &readSlotNbs[X_FEEDBACK_MSL_SLOT_IDX]))) {
    DSA_EXT_DIAG(err, devices->x);
    goto _error;
}

//Create a slot and link it to ML0 (position feedback) of axis Y
//User callback reads this slot and put the read value in internal memory
if ((err = dsa_create_rtv_read(devices->y, DMD_TYP_MONITOR_INT64, 0, 0, &yFeedback))) {
    DSA_EXT_DIAG(err, devices->y);
    goto _error;
}
//Retrieve the underlaying slot object to be able to retrive the low layer slot number
if ((err = dsa_get_rtv_slot_of_rtv(yFeedback, &helpSlot))) {
    DSA_EXT_DIAG(err, devices->y);
    goto _error;
}
//Retrieve the low layer slot numbers to be able to use low-level ETB write function
if ((err = dsa_get_64bit_rtv_slot_nr(helpSlot, &readSlotNbs[Y_FEEDBACK_LSL_SLOT_IDX],
                                     &readSlotNbs[Y_FEEDBACK_MSL_SLOT_IDX]))) {
    DSA_EXT_DIAG(err, devices->y);
    goto _error;
}

//Create a slot and link it to M63 (status) of axis X
//User callback reads this slot and put the read value in internal memory
if ((err = dsa_create_rtv_read(devices->x, DMD_TYP_MONITOR_INT32, 63, 0, &xStatus))) {
    DSA_EXT_DIAG(err, devices->x);
    goto _error;
}
//Retrieve the underlaying slot object to be able to retrive the low layer slot number
if ((err = dsa_get_rtv_slot_of_rtv(xStatus, &helpSlot))) {
    DSA_EXT_DIAG(err, devices->master);
    goto _error;
}
```

```
//Retrieve the low layer slot numbers to be able to use low-level ETB write function
if ((err = dsa_get_32bit_rtv_slot_nr(helpSlot, &readSlotNbs[X_STATUS_SLOT_IDX]))) {
   DSA_EXT_DIAG(err, devices->master);
   goto _error;
}

//Create a slot and link it to M63 (status) of axis Y
//User callback reads this slot and put the read value in internal memory
if ((err = dsa_create_rtv_read(devices->y, DMD_TYP_MONITOR_INT32, 63, 0, &yStatus))) {
   DSA_EXT_DIAG(err, devices->y);
   goto _error;
}
//Retrieve the underlaying slot object to be able to retrive the low layer slot number
if ((err = dsa_get_rtv_slot_of_rtv(yStatus, &helpSlot))) {
   DSA_EXT_DIAG(err, devices->master);
   goto _error;
}

//Retrieve the low layer slot numbers to be able to use low-level ETB write function
if ((err = dsa_get_32bit_rtv_slot_nr(helpSlot, &readSlotNbs[Y_STATUS_SLOT_IDX]))) {
   DSA_EXT_DIAG(err, devices->master);
   goto _error;
}
```

Computation of trajectory setpoints

The trajectory generated in this example is a simple circle. The corresponding X and Y positions are stored in a 2D table. The table size depends on the rate at which the application sends the setpoints to the Controller (for an update rate of 10 kHz, i.e. 1 setpoint every 100 µs, a total of 25000 setpoints are stored).

Remark: The trajectory computation is included in the source code that is provided with the EDI distribution package.

User callback

For this application example, the AccurET controller must be configured for the external position reference mode (ITP = 1). In this mode, the position reference is not given by the Controller's internal setpoint generator, but comes from a register which can be updated via RTV. In the user callback, the successive trajectory setpoints are written to the corresponding RTV slots that will update the Controller's position reference register.

The user callback executes two main tasks:
- Reads the RTVs containing the status and position measurements of the Controller.
- Writes to the RTVs the trajectory setpoints to update the Controller's position reference.

The first task is always executed at each cycle, while the second one is only executed when the main application has enabled it.

As presented in sections §14.2.3.1 and §14.2.3.2, RTVs must be read before 60 µs and written after 17 µs. Therefore, the user callback first reads the RTVs, then prepares the values to be written to the RTVs and waits until the time counter reaches 17 µs before writing them.

The user callback is also able to generate errors and signal these to the main application. These are the errors which are detected:

```
RTV_READ_SYNCHRO_ERROR = -1,          // The callback has read real-time values not in
                                      // imparted time

CALLBACK_DURATION_EXCEEDED = -2,      // The callback duration has exceeded the imparted
                                      // time and is no
```

```
//-----------------------------------------------------------------------------
// This is the function which will be called at each PCI IRQ. The devices have prealably
// set in ITP mode 1 (position reference provided by RTV slots)
// This function:
// - read the RTV values containing the devices status and devices actual position.
// - get the next trajectory's point to go to
// - if requested by min application
//      - waits for the time where writing RTV values are safe (17 us after TransnET IRQ)
//       - write the RTV values in the programmed slots.
// - store the read RTV values (status & actual position) in internal memory
// - compute the callback duration to allow generating an error if the duration exceeds
// the imparted time
// - compute the minimal callback start time to allow to fine tune the callback timing
// Refer to EDI4 RTX realtime application documentation
// - if an error is detected (read real-time values not done in imparted time, or
// callback duration exceeded), it generates an event signaling the error to the main
// application
//-----------------------------------------------------------------------------

static void _user_callback(DSA_MASTER *master, int nr,
                           int nb_read, DSA_RTV_DATA **read_rtv,
                           int nb_write, DSA_RTV_DATA **write_rtv, ETB *etb)
{
   int read_err = 0;
   int write_err = 0;
   S_POINT p;
   eint64 start_time_us, callback_duration_us;//Used to compute the duration
                                       //of the callback with OS time function
   double fpga_start_time_us;          //Used to compute the minimal start time
                                       //of the callback allowing a fine tuning

   //get the start time of callback (from PCI IRQ).
   //This take ~2 us and should be removed once callback has been fine tuned
   fpga_start_time_us = _get_FPGA_time(etb) * FPGA_TIME_TO_US;

   //get the start time using OS timing
   start_time_us = _get_time_us();

   //Compute the minimal start time (for fine tuning of callback
   if (realtime_callback_min_start_time_us > fpga_start_time_us)
   realtime_callback_min_start_time_us = fpga_start_time_us;

   //Read the real-time values containing status and position feedbacks.
   //Calling the function returns
   //an error if the reading cannot be done in the same TransnET cycle
   read_err = etb_read_slot_array(etb, NB_32BITS_READ_SLOTS, readSlotNbs, readSlotValps);

   //If required by main application, execute the trajectory
   if (trajectory.execute) {
      if (_get_next_point(&p) == -1) {
         trajectory.execute = FALSE;
         _trig_trajectory_end();
      }
      else {
         //Prepare the array of real-time values
         writeSlotVals[X_REFERENCE_LSL_SLOT_IDX] = LOWDWORD64(p.xi);
         writeSlotVals[X_REFERENCE_MSL_SLOT_IDX] = HIGHDWORD64(p.xi);
         writeSlotVals[Y_REFERENCE_LSL_SLOT_IDX] = LOWDWORD64(p.yi);
         writeSlotVals[Y_REFERENCE_MSL_SLOT_IDX] = HIGHDWORD64(p.yi);
         //Wait that the correct time to write the real-time values
         _wait_transnet_write_time(etb, 17 - USER_CALLBACK_DELAY);
         //Writes the real-time values
         etb_write_slot_array(etb, NB_32BITS_WRITE_SLOTS, writeSlotNbs,
         writeSlotVals);
      }

   }
```

```
   //No read error has been detected
   if (!read_err) {
      status_internal_update(readSlotVals[X_STATUS_SLOT_IDX],
readSlotVals[Y_STATUS_SLOT_IDX],
      readSlotVals[X_FEEDBACK_LSL_SLOT_IDX],
      readSlotVals[X_FEEDBACK_MSL_SLOT_IDX],
      readSlotVals[Y_FEEDBACK_LSL_SLOT_IDX],
      readSlotVals[Y_FEEDBACK_MSL_SLOT_IDX]);
   }
   //A read error has been detected => signal it to main application
   else {
      trajectory.execute = FALSE;
      trajectory.callback_error = RTV_READ_SYNCHRO_ERROR;
      _trig_callback_error();
   }
   //Compute the duration of the callback and generate an error if the imparted time is
   //exceeded
   callback_duration_us = _get_time_us() - start_time_us;
   if (callback_duration_us > USER_CALLBACK_RATE_US) {
      trajectory.execute = FALSE;
      trajectory.callback_error = CALLBACK_DURATION_EXCEEDED;
      _trig_callback_error();
   }
   return;

}
```

User callback start and stop

The user callback is enabled using the EDI function `dsa_start_delayed_rtv_handler`, which also configures the UltimET motion controller to generate a PCI/PCIe IRQ at a defined frequency and offset with respect to TransnET's IRQ.
The user must ensure that the execution of user callback does not overflow to the next cycle. Theoretically, the user callback execution could last up to 100 µs, but this is not possible because of the various latencies. ETEL recommends measuring the system latencies to determine the maximum execution duration feasible, applying a safety margin. Also, the user should monitor the time spent in the user callback and generate an error if it overflows the specified maximum.

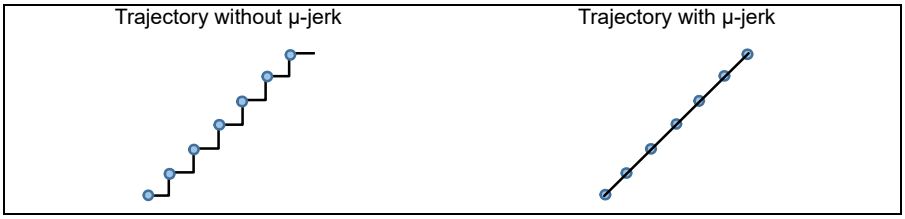
```
//-----------------------------------------------------------------------------------------
// Start the realtime callback
// The realtime callback is set with:
// - a delay of 0 (PCI IRQ not delayed from TransnET IRQ)
// - a frequency which depends of the OS:
//   RTX: 10 KHz (1 point each 100 us (TransnET frequency)
//   Windows: 40 Hz (1 point each 25 ms
// The realtime callback waits for the synchronization flag to execute the
// trajectory
// After this function, the callback will only read the RTV values and store them
// in internal memory
// The application will start the trajectory execution once the devices will be
// set in
// ITP=1 (position reference mode)
//-----------------------------------------------------------------------------------------
int realtime_callback_start()
{
   int err = 0;
   ETB *etb = NULL;
   if ((err = dsa_get_etb_bus(devices->master, &etb))) {
      DSA_DIAG(err, devices->master);
      goto _error;
   }
   printf("Start user callback\n");
   trajectory.execute = FALSE;
   trajectory.callback_error = NO_CALLBACK_ERROR;
```

```
   if ((err = dsa_start_delayed_rtv_handler(devices->master, 0,
(int)(USER_CALLBACK_RATE_US / 100.0),
                                            USER_CALLBACK_DELAY, _user_callback, 0,
                                            NULL, 0, NULL,
                                            etb))) {
      DSA_DIAG(err, devices->master);
      goto _error;
   }
_error:
   return err;
}

//-----------------------------------------------------------------------------
// Stop the real-time callback
//-----------------------------------------------------------------------------
int realtime_callback_stop()
{
   int err = 0;
   printf("Stop user callback\n");

   trajectory.execute = FALSE;
   if ((err = dsa_stop_rtv_handler(devices->master, 0))) {
      DSA_DIAG(err, devices->master);
      goto _error;
   }
_error:
   return err;
}
```

Enabling and disabling the real-time trajectory execution

Before it can enable the dispatching of trajectory setpoints to the Controller, the application must realize the following initialization steps:

- Move the axes to the first trajectory setpoint.
- Initialize the RTV with this first position to avoid a too big position step leading to a tracking error when the Controller's external position reference mode is enabled.
- Set the Controller's µ-jerk to linearize the position between two setpoints. The µ-jerk must be set equal to the duration between two setpoints (i.e. 100 µs).



- Enable the Controller's external position reference mode.

Finally, set the flag to enable the user callback to dispatch a trajectory setpoint per cycle.

```
//-----------------------------------------------------------------------------
// Start the realtime trajectory execution
// - Program the micro jerk of the devices. The micro jerk is set to the callback rate.
//   This forces the device to linearize the position between provided trajectory points
// - Initialize the reference realtime value slots with the first point of the trajectory.
//   This is requested, the axes would take their reference position from realtime value /
/ slot as soon as they would be set in ITP=1
// - Set the devices in ITP=1 mode (position reference from specified RTV slot)
// - Start the trajectory execution by setting the flag which tells the real-time callback
//   to send the trajectory points
//-----------------------------------------------------------------------------
int realtime_trajectory_start()
{
   int err = 0;
```

```
    S_POINT* initial_position = realtime_trajectory_get_initial_position();
    ETB *etb = NULL;

    printf("Start real-time trajectory\n");

    if ((err = devices_set_micro_jerk(USER_CALLBACK_RATE_US / 1000000.0))) {
       goto _error;
    }
    if ((err = dsa_get_etb_bus(devices->master, &etb))) {
       DSA_EXT_DIAG(err, devices->master);
       goto _error;
    }

    writeSlotVals[X_REFERENCE_LSL_SLOT_IDX] = LOWDWORD64(initial_position->xi);
    writeSlotVals[X_REFERENCE_MSL_SLOT_IDX] = HIGHDWORD64(initial_position->xi);
    writeSlotVals[Y_REFERENCE_LSL_SLOT_IDX] = LOWDWORD64(initial_position->yi);
    writeSlotVals[Y_REFERENCE_MSL_SLOT_IDX] = HIGHDWORD64(initial_position->yi);
    if ((err = etb_write_slot_array(etb, NB_32BITS_WRITE_SLOTS, writeSlotNbs,
writeSlotVals))) {
       ETB_ETCOM_DIAG(err, NULL, 0);
       goto _error;
    }
    if ((err = devices_enable_itp_mode(writeSlotNbs[X_REFERENCE_LSL_SLOT_IDX],
writeSlotNbs[X_REFERENCE_MSL_SLOT_IDX],
                                       writeSlotNbs[Y_REFERENCE_LSL_SLOT_IDX],
                                       writeSlotNbs[Y_REFERENCE_MSL_SLOT_IDX]))) {
        goto _error;
    }
    trajectory.execute = TRUE;
_error:
    return err;
}
```

To disable the trajectory execution, the application must:

- Unset the flag disabling the dispatch of trajectory setpoints from the user callback.
- Disable the Controller's external position reference mode.
- Reset the Controller's µ-jerk.

```
//-----------------------------------------------------------------------------------------
// Stop the realtime trajectory execution:
// - Reset the flag avoiding the real-time callback to send trajectory points
// - Disable position reference mode of the devices (ITP=0)
// - Set micro jerk to 0
//-----------------------------------------------------------------------------------------
int realtime_trajectory_stop()
{
   int err = 0;
   printf("Stop real-time trajectory\n");
   trajectory.execute = FALSE;

   if ((err = devices_disable_itp_mode())) {
      goto _error;
   }
   if ((err = devices_set_micro_jerk(0))) {
      goto _error;
   }

_error:
   return err;
}
```

14.4.3.2 Status management

This module contains the functions for managing the Controller's status and position feedback information which is collected by the user callback function and stored in an internal memory.

A low priority status handling task reads regularly this internal memory, converts the position measurements into ISO units and stores the data in a shared memory. Once finished, the task sets an event signaling that the shared memory has been updated and displays the status on the console.

Remark: In a real use case, the Windows application would get the status information from the shared memory and display it on the GUI.

Creating and starting the status task

```
//-----------------------------------------------------------------------------------------
// The function executed by status thread.
// This function:
// - get the devices position feedback (stored by realtime task in INCREMENT) and convert
// them in ISO
// - get the devices status (stored by realtime task)
// - write these values in shared memory which will be read and displayed by main
// application
// - In this example, we display the status in this task.
// - In a "normal" application, this should be done in a Windows task displaying this on
// a nice windows panel. Therefore, the shared memory is protected by a Critical section
// allowing this.
// Moreover, the update of the shared memory is signaled by an event which the windows
// application could wait for. In our example, nobody waits for this event.
//-----------------------------------------------------------------------------------------
static int _status_thread_fct(void *params)
{
   S_DEVICES *devices = (S_DEVICES *)params;
   int err = 0;
   double xIsoFeedback;
   double yIsoFeedback;
   dword xStatus;
   dword yStatus;
   while (!stop) {
   if ((err = dsa_convert_int64_to_iso(devices->x, &xIsoFeedback,
internal_status.feedback.xi,          DMD_CONV_UPI))) {
      DSA_DIAG(err, devices->x);
      goto _error;
   }
   if ((err = dsa_convert_int64_to_iso(devices->y, &yIsoFeedback,
internal_status.feedback.yi,          DMD_CONV_UPI))) {
      DSA_DIAG(err, devices->y);
      goto _error;
   }
   xStatus = internal_status.xStatus;
   yStatus = internal_status.yStatus;
   //Write data to shared memory
   CRITICAL_ENTER(status_shm);
   status_shm.xFeedback = xIsoFeedback;
   status_shm.yFeedback = yIsoFeedback;
   status_shm.xStatus = xStatus;
   status_shm.yStatus = yStatus;
   CRITICAL_LEAVE(status_shm);
   //Signal that the shared memory has been updated
   AUTOEVENT_SET(status_shm.updated);
   _status_display();
   SLEEP(500);
   }
_error:
   return err;
}

//-----------------------------------------------------------------------------------------
// Start the status thread
//-----------------------------------------------------------------------------------------
int status_thread_start(S_DEVICES *devices)
```

```
{
   printf("Start status thread\n");
   memset(&internal_status, 0, sizeof(internal_status));
   memset(&status_shm, 0, sizeof(status_shm));
   CRITICAL_INIT(status_shm);
   AUTOEVENT_INIT(status_shm.updated, FALSE);

   THREAD_INIT_AND_NAME_PRIO(status_thread, "Status Thr", status_thread_fct,
                             (void*)devices,EDI_THREAD_PRIORITY_IDLE);
   if (status_thread == THREAD_INVALID) {
      printf("Unable to create Status thread\n");
      return -1;
   }
   return 0;
}
```

14.4.3.3 Application main

The main function executes:

- Initialization: creation of device objects, opening communication and initialization of Controllers (reset errors, power on, homing…).
- Initiating the user callback.
- Initiating the status task.
- In a loop (fixed number of iterations)
    - Enabling the real-time trajectory execution.
    - Waiting for the user callback to complete the trajectory execution.
    - Disabling the trajectory execution.
- After executing all iterations or in case an error is detected, closing communication, stopping all tasks and destroying all objects.

```
//---------------------------------------------------------------------------------------
// main
// - Initialize the devices and real-time object
// - start a status thread which will display the status & position of the axes
// - Execute the real-time trajectory curve several times
//---------------------------------------------------------------------------------------
int main(void)
{
   int err;
   int i;
   ebool waiting;
   S_DEVICES *devices;
   printf("-----------------------------------------------------------------\n");
   printf("--- ETEL realtime application demo   ------------------------\n");
   printf("-----------------------------------------------------------------\n");
   if ((err = devices_create())) {
      goto _error;
   }
   if ((err = devices_open())) {
      goto _error;
   }
   if ((err = devices_init())) {
      goto _error;
   }
   devices = devices_get_object();
   if ((err = realtime_create(devices))) {
      goto _error;
   }
   if ((err = realtime_callback_start())) {
      goto _error;
   }
   if ((err = status_thread_start(devices))) {
      goto _error;
   }
   for (i = 0; i < NB_LOOP; i++) {
      if ((err = _start_trajectory_execution())) {
```

```
            goto _error;
         }
         waiting = TRUE;
         while (waiting) {
            switch (_wait_events()) {
               case TRAJECTORY_ERROR:
                  printf("Trajectory error %d\n", realtime_callback_get_error_nb());
                  if ((err = _stop_trajectory_execution())) {
                     goto _error;
                  }
                  waiting = FALSE;
                  break;
               case TRAJECTORY_FINISHED:
                  printf("Trajectory successfully ended\n");
                  if ((err = _stop_trajectory_execution())) {
                     goto _error;
                  }
                  printf("Main go to take a small nap...\n");
                  SLEEP(5000);
                  waiting = FALSE;
                  break;
               default:
                  printf("Event Wait error\n");
                  err = -1;
                  waiting = FALSE;
                  break;
            }
         }
      }

_error:
   _stop_trajectory_execution();
   realtime_callback_stop();
   status_thread_stop();
   realtime_destroy();
   devices_close();
   devices_destroy();
   if (err)
      printf("Ended with error %d\n", err);
   else
      printf("Ended successfully\n");

}
```

15 Appendixes

15.1 DSA library functions

15.1.1 Functions to send commands

ETEL syntax	Command number	EDI function
ASG	235	
ASM	230	
ASP	234	
ASR	231	dsa_free_all_transnET_slots_s
		dsa_unassign_slot_to_register_a
		dsa_unassign_slot_to_register_s
		dsa_assign_slot_to_register_s
		dsa_assign_slot_to_register_a
ASS	252	
ASW	232	dsa_assign_register_to_slot_s
		dsa_free_all_transnET_slots_s
		dsa_unassign_register_to_slot_a
		dsa_unassign_register_to_slot_s
		dsa_assign_register_to_slot_a
AUT	150	
AXI	109	
BRK	70	dsa_quick_stop_s
		dsa_quick_stop_a
CAL	68	
CEC	58	
CFGPGFIX	310	
CFGPGMOD	311	
CFGTRIGEXTPOS	312	
CH_BIT_REG32	90	dsa_modify_bits_int32_register_s
		dsa_modify_bits_int32_register_a
CHKDISTGRT	563	
CHKMD5	510	
CHKMD5SWAP	512	
CLRWAIT	180	
CLX	17	
CSH	21	
CTFOFF	301	
CTFON	300	
CTFRTV	302	
CWS	522	
DEV_REMAP	505	
DEV_TRAP_CLEAN_ALL	501	
DEV_TRAP_INSERT	500	
DEV_TRAP_SNIPPET_SELECT	503	
DEV_TRAP_WIND_EVENT	502	
DEV_UNLOCK	504	
DEV_WRITE	506	

ETEL syntax	Command number	EDI function
DFC	241	
DFCLD	210	
DIAGNOSE	499	
DIS	82	
DTC	250	
DWN	42	
DWNFW	216	
ECAT_ASW	229	
ECAT_ITP	117	
EDF	239	
EIOMAXURST	755	dsa_externalIO_reset_max_update_time_s
EIOSTA	750	dsa_externalIO_set_enable_cyclic_update_s
EIOWDEN	720	dsa_externalIO_enable_watchdog_s
		dsa_externalIO_disable_watchdog_s
EIOWDSTP	721	dsa_externalIO_stop_watchdog_s
EIOWDTIME	722	dsa_externalIO_set_watchdog_time_s
END	0	dsa_stop_sequence_a
		dsa_stop_sequence_in_thread_s
		dsa_stop_sequence_in_thread_a
		dsa_stop_customer_software_module_function_s
		dsa_stop_sequence_s
		dsa_stop_customer_software_module_function_a
ERR	80	
ERRORAUX	85	
ESMLOAD	221	
ESMUNLOAD	222	
FCOFF	301	dsa_force_control_dual_axes_off_a
		dsa_force_control_dual_axes_off_s
FCON	300	dsa_force_control_dual_axes_on_s
		dsa_force_control_dual_axes_on_a
FCRTV	302	
FGA	313	
FLASH_EXEC	507	
FLT	222	
FORCE_LIMIT	118	
FREESLOT	531	dsa_free_32bit_rtv_slot
		dsa_free_64bit_rtv_slot
		dsa_free_all_transnET_slots_s
FWD	520	
GETDINS	738	dsa_localIO_get_digital_input_state_s
GETDOUT	735	dsa_localIO_get_digital_output_s
GETEAINCS	746	dsa_externalIO_get_analog_input_converted_data_state_s
GETEAINRS	745	dsa_externalIO_get_analog_input_raw_data_state_s
GETEAOUTC	741	dsa_externalIO_get_analog_output_converted_data_s
GETEAOUTCS	743	dsa_externalIO_get_analog_output_converted_data_state_s
GETEAOUTR	740	dsa_externalIO_get_analog_output_raw_data_s
GETEAOUTRS	742	dsa_externalIO_get_analog_output_raw_data_state_s

ETEL syntax	Command number	EDI function
GETEDINS	765	dsa_externalIO_get_digital_input_state_s
GETEDOUT	760	dsa_externalIO_get_digital_output_s
GETEDOUTS	764	dsa_externalIO_get_digital_output_state_s
GETEREG	726	dsa_externalIO_get_modbus_register_s
GETMDINS	734	dsa_localIO_get_masked_digital_input_state_s
GETMDOUT	732	dsa_localIO_get_masked_digital_output_s
GETMEDINS	758	dsa_externalIO_get_masked_digital_input_state_s
GETMEDOUT	757	dsa_externalIO_get_masked_digital_output_s
GETMEDOUTS	766	dsa_externalIO_get_masked_digital_output_state_s
GETSLOT	530	dsa_get_64bit_rtv0_slot
		dsa_get_32bit_rtv0_slot
GFD	229	
GFS	228	dsa_upload_firmware_file_s
		dsa_upload_log_file_s
		dsa_upload_parameter_file_s
		dsa_upload_zipped_log_file_s
		dsa_upload_encrypted_log_file_s
		dsa_upload_file_s
GGA	312	
GLFI	215	
GZFS	218	
HLB	121	dsa_ipol_quick_stop_s
		dsa_quick_stop_a
		dsa_quick_stop_s
		dsa_ipol_quick_stop_a
HLO	119	dsa_quick_stop_s
		dsa_quick_stop_a
HLT	120	dsa_quick_stop_a
		dsa_ipol_quick_stop_s
		dsa_ipol_quick_stop_a
		dsa_quick_stop_s
IABSCOORDS	556	dsa_ipol_abs_coords_a
		dsa_ipol_abs_coords_s
IABSMODE	555	dsa_ipol_set_abs_mode_a
		dsa_ipol_set_abs_mode_s
IBEGIN	553	dsa_ipol_begin_a
		dsa_ipol_begin_s
IBRK	653	dsa_ipol_quick_stop_s
		dsa_ipol_quick_stop_a
ICCW	1041	dsa_ipol_circle_ccw_c2d_a
		dsa_ipol_circle_ccw_c2d_s
ICCWR	1027	dsa_ipol_circle_ccw_r2d_a
		dsa_ipol_circle_ccw_r2d_s
ICLRB	657	dsa_ipol_clear_buffer_s
		dsa_ipol_clear_buffer_a
ICONC	1030	dsa_ipol_begin_concatenation_s
		dsa_ipol_begin_concatenation_a

ETEL syntax	Command number	EDI function
ICONT	654	dsa_ipol_continue_a
		dsa_ipol_continue_s
ICW	1040	dsa_ipol_circle_cw_c2d_a
		dsa_ipol_circle_cw_c2d_s
ICWR	1026	dsa_ipol_circle_cw_r2d_a
		dsa_ipol_circle_cw_r2d_s
IEND	554	dsa_ipol_end_a
		dsa_ipol_end_s
IEQ	151	
IGE	156	
IGT	154	
ILE	155	
ILINE	1025	dsa_ipol_line_a
		dsa_ipol_line_2d_a
		dsa_ipol_line_2d_s
		dsa_ipol_line_s
ILKT	1032	dsa_ipol_lkt_a
		dsa_ipol_lkt_s
ILOCK	1044	dsa_ipol_lock_a
		dsa_ipol_lock_s
ILT	153	
IMARK	1039	dsa_ipol_mark_2param_s
		dsa_ipol_mark_a
		dsa_ipol_mark_2param_a
		dsa_ipol_mark_s
IMRES	1063	
IMROT	1056	dsa_ipol_rotate_matrix_a
		dsa_ipol_rotate_matrix_s
IMSCALE	1055	dsa_ipol_scale_matrix_2d_a
		dsa_ipol_scale_matrix_2d_s
		dsa_ipol_scale_matrix_a
		dsa_ipol_scale_matrix_s
IMSHEAR	1057	dsa_ipol_shear_matrix_a
		dsa_ipol_shear_matrix_s
IMTRANS	1054	dsa_ipol_translate_matrix_a
		dsa_ipol_translate_matrix_2d_s
		dsa_ipol_translate_matrix_2d_a
		dsa_ipol_translate_matrix_s
INCONC	1031	dsa_ipol_end_concatenation_a
		dsa_ipol_end_concatenation_s
IND	45	dsa_homing_start_s
		dsa_homing_start_a
INE	152	
INI	44	
INS	41	
IOCFGRST	595	
IOIPOLTEST	596	

ETEL syntax	Command number	EDI function
IOSHOWPINS	597	
IPT	1045	dsa_ipol_pt_a
		dsa_ipol_pt_s
IPVT	1028	dsa_ipol_pvt_s
		dsa_ipol_pvt_a
		dsa_ipol_pvt_reg_typ_s
		dsa_ipol_pvt_reg_typ_a
IPVTUPDATE	662	dsa_ipol_pvt_update_a
		dsa_ipol_pvt_update_s
ISET	552	dsa_ipol_begin_s
		dsa_ipol_begin_a
ISTP	656	dsa_ipol_quick_stop_a
		dsa_ipol_quick_stop_s
ITACC	1036	dsa_ipol_tan_acceleration_s
		dsa_ipol_tan_acceleration_a
ITANSPDMASK	1074	
ITDEC	1037	dsa_ipol_tan_deceleration_a
		dsa_ipol_tan_deceleration_s
ITJRT	1038	dsa_ipol_tan_jerk_time_s
		dsa_ipol_tan_jerk_time_a
ITP	116	
ITRIG	1042	
ITSPD	1035	dsa_ipol_tan_velocity_a
		dsa_ipol_tan_velocity_s
IULINE	1033	dsa_ipol_uline_time_a
		dsa_ipol_uline_s
		dsa_ipol_uline_time_s
		dsa_ipol_uline_2d_s
		dsa_ipol_uline_2d_a
		dsa_ipol_uline_time_2d_s
		dsa_ipol_uline_time_2d_a
		dsa_ipol_uline_a
IUNLOCK	655	dsa_ipol_unlock_s
		dsa_ipol_unlock_a
IUNOCONC	1052	dsa_ipol_disable_uconcatenation_a
		dsa_ipol_disable_uconcatenation_s
IURELATIVE	1051	dsa_ipol_set_urelative_mode_s
		dsa_ipol_set_urelative_mode_a
IUSPDMASK	1053	dsa_ipol_uspeed_axis_mask_a
		dsa_ipol_uspeed_axis_mask_s
IUSPEED	1049	dsa_ipol_uspeed_a
		dsa_ipol_uspeed_s
IUTIME	1050	dsa_ipol_utime_s
		dsa_ipol_utime_a
IWTT	1029	
JBC	37	
JBS	36	

ETEL syntax	Command number	EDI function
JEQ	137	
JGT	138	
JLT	136	
JMP	26	dsa_execute_sequence_in_thread_s
		dsa_execute_sequence_s
		dsa_start_customer_software_module_function_a
		dsa_execute_sequence_in_thread_a
		dsa_start_customer_software_module_function_s
		dsa_execute_sequence_a
JNE	139	
MAGCTRL	160	
MAM	199	
MCS	200	
MCT	193	
MD5_FSUM	511	
MD5_SUM	510	
MDA	195	
MDT	192	
MMF	562	
MMO	194	
MSI	190	
MSR	196	
MSV	191	
MTP	197	
MTU	198	
MVE	60	dsa_start_profiled_movement_ex_a
		dsa_start_profiled_movement_s
		dsa_start_profiled_movement_a
		dsa_start_profiled_movement_ex_s
MVEDOF	1539	dsa_move_ndof_s
		dsa_set_all_axes_motion_profile_data_and_move_s
MVETILT	480	dsa_start_tilt_movement_s
		dsa_start_tilt_movement_a
NEW	78	dsa_default_parameters_a
		dsa_default_parameters_s
NEWFC	305	dsa_force_control_new_s
		dsa_force_control_new_a
		dsa_force_control_new_ex1_s
		dsa_force_control_new_ex2_s
		dsa_force_control_new_ex2_a
		dsa_force_control_new_ex1_a
NOG	149	
OFFSETAUX	24	
OFFSETSEC	23	
PARAMETERSWITCH	1536	dsa_activate_parameters_s
PBK	221	
PCT	752	

ETEL syntax	Command number	EDI function
POP	34	
POSCAPT	211	
PTS	76	
PWR	124	dsa_quick_stop_a
		dsa_power_on_a
		dsa_quick_stop_s
		dsa_power_on_s
		dsa_power_off_a
		dsa_power_off_s
PWR_MUL	83	
RCT	754	dsa_externalIO_reset_client_communication_s
READCYCLIC	516	
READJUSTPOS	86	
RES	49	dsa_load_parameters_a
		dsa_load_parameters_s
RESETFC	304	dsa_force_control_reset_ex3_a
		dsa_force_control_reset_ex3_s
		dsa_force_control_reset_ex2_a
		dsa_force_control_reset_ex2_s
		dsa_force_control_reset_ex1_a
		dsa_force_control_reset_ex1_s
		dsa_force_control_reset_a
		dsa_force_control_reset_s
RET	69	
RIC	753	dsa_externalIO_reset_io_cycle_count_s
RMF	219	dsa_remove_customer_software_module_s
		dsa_remove_parameter_file_s
		dsa_remove_etel_software_module_s
RMVE	62	dsa_start_relative_profiled_movement_a
		dsa_start_relative_profiled_movement_ex_s
		dsa_start_relative_profiled_movement_s
		dsa_start_relative_profiled_movement_ex_a
RMVETILT	481	dsa_start_relative_tilt_movement_s
		dsa_start_relative_tilt_movement_a
RSD	88	
RSH	600	
RST	79	dsa_reset_error_with_check_ex_a
		dsa_reset_error_with_check_a
		dsa_reset_error_with_check_ex_s
		dsa_reset_error_with_check_s
		dsa_reset_error_s
		dsa_reset_error_ex_s
		dsa_reset_error_ex_a
		dsa_reset_error_a
RSTDOUT	737	dsa_localIO_reset_digital_output_s
RSTEDOUT	762	dsa_externalIO_reset_digital_output_s
RSU	601	

ETEL syntax	Command number	EDI function
RTVDOUT	229	
RTVFFWD	228	
RTVQUIET	227	
SAF	63	dsa_set_advanced_filter_a
		dsa_set_advanced_filter_s
SAFRX	489	dsa_set_rx_advanced_filter_s
		dsa_set_rx_advanced_filter_a
SAFRY	487	dsa_set_tilt_advanced_filter_s
		dsa_set_tilt_advanced_filter_a
		dsa_set_ry_advanced_filter_a
		dsa_set_ry_advanced_filter_s
SAV	48	dsa_save_parameters_a
		dsa_save_parameters_s
SCI	102	
SDF	240	
SDP	248	
SDS	245	
SDSF	226	dsa_download_file_s
		dsa_download_customer_software_module_s
SEQBKPALL	142	dsa_debug_sequence_enable_breakpoint_everywhere_s
		dsa_debug_sequence_enable_breakpoint_everywhere_a
SEQBRKTHR	143	
SEQCONT	140	dsa_debug_sequence_continue_a
		dsa_debug_sequence_continue_s
SEQDBGEND	143	
SEQENBKP	141	dsa_debug_sequence_enable_breakpoint_at_a
		dsa_debug_sequence_enable_breakpoint_at_s
SET	22	dsa_set_user_position_s
		dsa_set_user_position_a
SET_RANGE	126	
SETDOUT	736	dsa_localIO_set_digital_output_s
SETEAOUTC	731	dsa_externalIO_set_analog_output_converted_data_s
SETEAOUTR	730	dsa_externalIO_set_analog_output_raw_data_s
SETEDOUT	761	dsa_externalIO_set_digital_output_s
SETEREG	725	dsa_externalIO_set_modbus_register_s
SETFC	303	dsa_force_control_set_ex3_s
		dsa_force_control_set_ex3_a
		dsa_force_control_set_ex2_a
		dsa_force_control_set_ex2_s
		dsa_force_control_set_ex1_a
		dsa_force_control_set_ex1_s
		dsa_force_control_set_a
		dsa_force_control_set_ex4_s
		dsa_force_control_set_ex4_a
		dsa_force_control_set_s

ETEL syntax	Command number	EDI function
SETFCI	306	dsa_force_control_set_immediate_ex1_s
		dsa_force_control_set_immediate_s
		dsa_force_control_set_immediate_ex2_s
		dsa_force_control_set_immediate_ex2_a
		dsa_force_control_set_immediate_a
		dsa_force_control_set_immediate_ex1_a
SETLOGLEVEL	535	
SETMDOUT	733	dsa_localIO_apply_mask_digital_output_s
SETMEDOUT	763	dsa_externalIO_apply_mask_digital_output_s
SETRTV	532	
SETTILT	483	dsa_set_tilt_user_position_a
		dsa_set_tilt_user_position_s
SETVPAR	1537	dsa_set_vector_parameter_int64_s
		dsa_set_vector_parameter_s
SETVPARM	1538	dsa_set_matrix_parameter_s
		dsa_set_vector_parameter_mask_int64_s
		dsa_set_matrix_parameter_int64_s
		dsa_set_matrix_parameter_mask_s
		dsa_set_matrix_parameter_mask_int64_s
		dsa_set_vector_parameter_mask_s
SLS	46	dsa_search_limit_stroke_s
		dsa_search_limit_stroke_a
SLT	27	
SMP	61	dsa_set_profiled_movement_ex_s
		dsa_set_profiled_movement_s
		dsa_set_profiled_movement_ex_a
		dsa_set_profiled_movement_a
SMPDOF	1540	dsa_set_all_axes_motion_profile_data_s
SPG	105	
SPI2	220	
SPIABORT	579	
SPIBURSTCFG	586	dsa_start_burst_transfer_s
		dsa_start_burst_transfer_a
SPICON	580	dsa_SPI_connect_a
		dsa_SPI_connect_s
SPICYCLICCFG	585	dsa_config_cyclic_transfer_s
		dsa_config_cyclic_transfer_a
SPIDIS	581	dsa_SPI_disconnect_a
		dsa_SPI_disconnect_s
SPIREAD	582	dsa_SPI_read_s
		dsa_SPI_read_a
SPIRST	589	
SPIRW	584	dsa_SPI_read_write_s
		dsa_SPI_read_write_a
SPIWRITE	583	dsa_SPI_write_s
		dsa_SPI_write_a

ETEL syntax	Command number	EDI function
SSR	127	dsa_set_several_iso_registers_a
		dsa_set_several_iso_registers_s
		dsa_set_several_registers_a
		dsa_set_several_registers_s
STA	25	dsa_start_movement_s
		dsa_change_setpoint_a
		dsa_start_movement_a
		dsa_change_setpoint_s
		dsa_new_setpoint_s
		dsa_new_setpoint_a
STARTBURST	587	dsa_start_burst_transfer_s
		dsa_start_burst_transfer_a
STARTCSM	224	dsa_start_customer_software_module_a
		dsa_start_customer_software_module_s
STARTCYCLIC	590	dsa_start_cyclic_transfer_handler
STE_ABS	129	dsa_step_motion_s
		dsa_step_motion_a
STE_ADD	114	
STE_SUB	115	
STETILT_ABS	484	
STETILT_ADD	485	
STETILT_SUB	486	
STF	244	
STI	33	
STOPCSM	225	dsa_stop_customer_software_module_a
		dsa_stop_customer_software_module_s
STOPCYCLIC	591	dsa_stop_cyclic_transfer_handler
STP	18	dsa_quick_stop_a
		dsa_quick_stop_s
STR	237	
STV	20	dsa_set_parameters_version_s
		dsa_set_parameters_version_a
SUF	242	
SUMAG	239	dsa_start_upload_memory_s
SUMETA	253	dsa_start_upload_metadata_s
SUP	249	dsa_start_upload_register_s
SUS	246	dsa_start_upload_sequence_s
		dsa_upload_sequence_s
SUSF	227	dsa_upload_firmware_file_s
		dsa_upload_parameter_file_s
		dsa_upload_file_s
		dsa_upload_encrypted_log_file_s
		dsa_upload_zipped_log_file_s
		dsa_upload_log_file_s
SUT	247	dsa_start_upload_trace_s
SUZSF	217	
TCPERR	693	

ETEL syntax	Command number	EDI function
TEST	767	
TIMING	110	
TMA	236	
TNS	77	
TOUCHPROBE	212	
TRANCRC	694	
TRANSYNC	695	
TRE	104	dsa_trigger_disable_a
		dsa_trigger_disable_s
		dsa_trigger_enable_ex_a
		dsa_trigger_enable_a
		dsa_trigger_enable_s
		dsa_trigger_enable_ex_s
TRESET	540	
TRF	233	
TRM	103	dsa_trigger_enable_s
		dsa_trigger_disable_a
		dsa_trigger_continuous_last_event_loaded_s
		dsa_trigger_continuous_last_event_loaded_a
		dsa_trigger_disable_s
		dsa_trigger_enable_ex_a
		dsa_trigger_enable_a
		dsa_trigger_enable_ex_s
TRM2D	108	dsa_trigger2d_continuous_last_event_loaded_a
		dsa_trigger2d_enable_ex_s
		dsa_trigger2d_continuous_last_event_loaded_s
		dsa_trigger2d_enable_s
		dsa_trigger2d_enable_a
		dsa_trigger2d_disable_a
		dsa_trigger2d_disable_s
		dsa_trigger2d_enable_ex_a
TRR	106	dsa_trigger_init_a
		dsa_trigger_init_s
TRS	107	dsa_trigger_status_enable_a
		dsa_trigger_status_disable_a
		dsa_trigger_status_enable_s
		dsa_trigger_status_disable_s
TST	74	
TTL_EMU	185	dsa_ttl_encoder_emulator_enable_s
		dsa_ttl_encoder_emulator_enable_a
UATEST	255	
UDPERR	692	
UDS	251	dsa_upload_data_s
		dsa_upload_sequence_s
UFC	243	
UGO	220	

ETEL syntax	Command number	EDI function
UST	30	dsa_user_stretch_disable_s
		dsa_user_stretch_enable_s
		dsa_user_stretch_enable_a
		dsa_user_stretch_disable_a
VALPOS	130	
WAB	13	
WAITBURST	588	dsa_start_burst_transfer_s
		dsa_start_burst_transfer_a
WAITCSM	238	dsa_wait_customer_software_module_s
		dsa_wait_customer_software_module_a
WBC	54	dsa_wait_bit_clear_a
		dsa_wait_bit_clear_s
WBS	55	dsa_wait_bit_set_s
		dsa_wait_bit_set_a
WPG	53	
WPL	52	
WRST	81	
WSBC	515	
WSBS	514	
WSG	57	dsa_wait_sgn_register_greater_a
		dsa_wait_sgn_register_greater_s
WSL	56	dsa_wait_sgn_register_lower_s
		dsa_wait_sgn_register_lower_a
WTD	15	dsa_acquisition_continuous_wait_available_values_s
		dsa_acquisition_continuous_wait_available_values_a
WTF	14	dsa_force_control_wait_a
		dsa_force_control_wait_s
WTK	513	dsa_ipol_wait_mark_a
		dsa_ipol_wait_mark_s
WTM	8	dsa_wait_movement_s
		dsa_wait_movement_a
WTP	9	dsa_wait_position_a
		dsa_wait_position_s
WTS	12	
WTT	10	dsa_wait_time_s
		dsa_wait_time_a
WTW	11	dsa_wait_window_a
		dsa_wait_window_s
WTWZXT	488	dsa_wait_window_zxt_a
		dsa_wait_window_zxt_s

ETEL syntax	Command number	EDI function
XYY_ABS	123	dsa_set_iso_register_s
		dsa_set_iso_register_a
		dsa_set_register_int64_s
		dsa_set_register_a
		dsa_set_register_int32_s
		dsa_set_register_float32_s
		dsa_set_register_float64_s
		dsa_set_register_s
		dsa_set_register_int32_a
		dsa_set_register_int64_a
		dsa_set_register_float32_a
		dsa_set_register_float64_a
XYY_ADD	91	
XYY_AND	95	dsa_register_int32_logical_and_s
		dsa_register_int64_logical_and_a
		dsa_register_int64_logical_and_s
		dsa_register_int32_logical_and_a
XYY_AND_NOT	97	dsa_register_int32_logical_and_not_a
		dsa_register_int64_logical_and_not_a
		dsa_register_int64_logical_and_not_s
		dsa_register_int32_logical_and_not_s
XYY_CONV	122	
XYY_DIV	94	
XYY_MODULO	101	
XYY_MUL	93	
XYY_NOT	174	
XYY_ORL	96	dsa_register_int32_logical_or_a
		dsa_register_int64_logical_or_a
		dsa_register_int64_logical_or_s
		dsa_register_int32_logical_or_s
XYY_ORL_NOT	98	dsa_register_int64_logical_or_not_a
		dsa_register_int32_logical_or_not_a
		dsa_register_int64_logical_or_not_s
		dsa_register_int32_logical_or_not_s
XYY_SET_MULTI	125	
XYY_SHL	173	
XYY_SHR	172	
XYY_SUB	92	
XYY_XOR	99	
XYY_XOR_NOT	100	
YLD	205	

ETEL syntax	Command number	EDI function
ZFT	203	dsa_sync_trace_force_trigger_s
		dsa_sync_trace_force_trigger_a
		dsa_acquisition_acquire_a
		dsa_acquisition_acquire_s
		dsa_acquisition_acquire_s
		dsa_acquisition_continuous_acquire
		dsa_sync_trace_enable_a
		dsa_sync_trace_enable_s
		dsa_acquisition_acquire_a

15.1.2 Functions for the reading and the writing of the registers

Registers	Alias	EDI read function	EDI write function
C107 or CF107		dsa_get_analog_output_s	dsa_set_analog_output_s
C30	XSRT	dsa_get_mon_source_type_s	dsa_set_mon_source_type_s
C31	XSRI	dsa_get_mon_source_index_s	dsa_set_mon_source_index_s
C359:1		dsa_trigger_get_fdout_mask_s	dsa_trigger_set_fdout_mask_s
C5	FDOUT	dsa_get_fast_digital_output_s	dsa_set_fast_digital_output_s
C6	XDOUT	dsa_get_x_digital_output_s	dsa_set_x_digital_output_s
C7:0		dsa_get_x_analog_output_1_s	dsa_set_x_analog_output_1_s
C7:1		dsa_get_x_analog_output_2_s	dsa_set_x_analog_output_2_s
C7:2		dsa_get_x_analog_output_3_s	dsa_set_x_analog_output_3_s
C7:3		dsa_get_x_analog_output_4_s	dsa_set_x_analog_output_4_s
K11		dsa_get_ttl_speed_filter_s	dsa_set_ttl_speed_filter_s
K164		dsa_get_syncro_start_timeout_s	dsa_set_syncro_start_timeout_s
K171	DOUT	dsa_get_digital_output_s	dsa_set_digital_output_s
K190		dsa_force_control_get_default_force_duration_s	dsa_force_control_set_default_force_duration_s
K198		dsa_get_indirect_register_idx_s	dsa_set_indirect_register_idx_s
K201	MMC	dsa_get_concatenated_mvt_s	dsa_set_concatenated_mvt_s
K202	MMD	dsa_get_profile_type_s	dsa_set_profile_type_s
K203	LTN	dsa_get_mvt_lkt_number_s	dsa_set_mvt_lkt_number_s
K204	LTI	dsa_get_mvt_lkt_time_s	dsa_set_mvt_lkt_time_s
K213	JRT	dsa_get_jerk_time_s	dsa_set_jerk_time_s
K220		dsa_get_ctrl_source_type_s	dsa_set_ctrl_source_type_s
K220:sidx		dsa_get_ctrl_source_type_ex_s	dsa_set_ctrl_source_type_ex_s
K221		dsa_get_ctrl_source_index_s	dsa_set_ctrl_source_index_s
K221:sidx		dsa_get_ctrl_source_index_ex_s	dsa_set_ctrl_source_index_ex_s
K223		dsa_get_ctrl_offset_s	dsa_set_ctrl_offset_s
K223:sidx		dsa_get_ctrl_offset_ex_s	dsa_set_ctrl_offset_ex_s
K239		dsa_get_motor_kt_factor_s	dsa_set_motor_kt_factor_s
K30		dsa_get_following_error_window_s	dsa_set_following_error_window_s
K302			dsa_force_control_set_enable_s
K31		dsa_get_velocity_error_limit_s	dsa_set_velocity_error_limit_s
K32		dsa_get_switch_limit_mode_s	dsa_set_switch_limit_mode_s
K320+(combi):0		dsa_trigger_get_combi_dout_mask_s	dsa_trigger_set_combi_dout_mask_s
K320+(combi):1		dsa_trigger_get_combi_fdout_mask_s	dsa_trigger_set_combi_fdout_mask_s
K320+(combi):3		dsa_trigger_get_combi_event_counter_group_s	dsa_trigger_set_combi_event_counter_group_s
K320+(combi):7		dsa_trigger_get_combi_user_status_mask_s	dsa_trigger_set_combi_user_status_mask_s
K33		dsa_get_enable_input_mode_s	dsa_set_enable_input_mode_s
K336		dsa_trigger_get_position_type_s	dsa_trigger_set_position_type_s

Registers	Alias	EDI read function	EDI write function
K337:0		dsa_trigger_get_continuous_nb_free_event_s	dsa_trigger_set_continuous_nb_free_event_s
K337:1		dsa_trigger_get_continuous_user_status_bit_s	dsa_trigger_set_continuous_user_status_bit_s
K338		dsa_trigger2d_get_box_tolerance_increment_s	dsa_trigger2d_set_box_tolerance_increment_s
K338		dsa_trigger2d_get_box_tolerance_s	dsa_trigger2d_set_box_tolerance_increment_s
K339:(pulse_-generator)		dsa_trigger_get_pulse_generator_period_s	dsa_trigger_set_pulse_generator_period_s
K340+(pulse_-generator):0		dsa_trigger_get_pulse_generator_dout_mask_s	dsa_trigger_set_pulse_generator_dout_mask_s
K340+(pulse_-generator):1		dsa_trigger_get_pulse_generator_fdout_mask_s	dsa_trigger_set_pulse_generator_fdout_mask_s
K342:(pulse_-generator)		dsa_trigger_get_pulse_generator_delay_s	dsa_trigger_set_pulse_generator_delay_s
K343:(pulse_-generator)		dsa_trigger_get_pulse_generator_pulse_width_s	dsa_trigger_set_pulse_generator_pulse_width_s
K344:(pulse_-generator)		dsa_trigger_get_pulse_generator_interval_s	dsa_trigger_set_pulse_generator_interval_s
K345:(pulse_-generator)		dsa_trigger_get_pulse_generator_number_s	dsa_trigger_set_pulse_generator_number_s
K347		dsa_trigger_get_missed_event_detection_time-out_s	dsa_trigger_set_missed_event_detection_time-out_s
K348		dsa_trigger_get_missed_event_detection_toler-ance_increment_s	dsa_trigger_set_missed_event_detection_toler-ance_increment_s
K348		dsa_trigger_get_missed_event_detection_toler-ance_s	dsa_trigger_set_missed_event_detection_toler-ance_increment_s
K349		dsa_trigger_get_missed_event_detection_action_s	dsa_trigger_set_missed_event_detection_ac-tion_s
K353:(trigger_-feature)		dsa_trigger_get_source_register_type_s	dsa_trigger_set_source_register_type_s
K354:(trigger_-feature)		dsa_trigger_get_source_register_idx_s	dsa_trigger_set_source_register_idx_s
K355:(trigger_-feature)		dsa_trigger_get_source_register_sidx_s	dsa_trigger_set_source_register_sidx_s
K356:(trigger_-feature)		dsa_trigger_get_source_register_axis_s	dsa_trigger_set_source_register_axis_s
K357:0		dsa_get_error_dout_mask_s	dsa_set_error_dout_mask_s
K357:1		dsa_get_error_fdout_mask_s	dsa_set_error_fdout_mask_s
K359		dsa_trigger_get_dout_mask_s	dsa_trigger_set_dout_mask_s
K36	MODESL	dsa_get_profile_limit_mode_s	dsa_set_profile_limit_mode_s
K360		dsa_trigger_get_position_mean_filter_s	dsa_trigger_set_position_mean_filter_s
K360:1		dsa_trigger_get_position_compensation_s	dsa_trigger_set_position_compensation_s
K38	TIMEW	dsa_get_position_window_time_s	dsa_set_position_window_time_s
K39	POSW	dsa_get_position_window_s	dsa_set_position_window_s
K40	HMODE	dsa_get_homing_method_s	dsa_set_homing_method_s
K52		dsa_get_homing_fine_tuning_mode_s	dsa_set_homing_fine_tuning_mode_s
K53		dsa_get_homing_fine_tuning_value_s	dsa_set_homing_fine_tuning_value_s
K530	ISPDRATE		dsa_ipol_set_velocity_rate_s
K56		dsa_get_motor_phase_correction_s	dsa_set_motor_phase_correction_s
K61		dsa_get_drive_control_mode_s	dsa_set_drive_control_mode_s
K66		dsa_get_display_mode_s	dsa_set_display_mode_s
K68		dsa_get_encoder_inversion_s	dsa_set_encoder_inversion_s
K7		dsa_get_pl_integrator_mode_s	dsa_set_pl_integrator_mode_s
K70		dsa_get_encoder_phase_1_offset_s	dsa_set_encoder_phase_1_offset_s
K700	ILKTCAM		dsa_ipol_set_lkt_speed_ratio_s
K71		dsa_get_encoder_phase_2_offset_s	dsa_set_encoder_phase_2_offset_s
K710	ILKTCYCLIC		dsa_ipol_set_lkt_cyclic_mode_s

Registers	Alias	EDI read function	EDI write function
K711	ILKTREL		dsa_ipol_set_lkt_relative_mode_s
K75		dsa_get_encoder_index_distance_s	dsa_set_encoder_index_distance_s
K90		dsa_get_init_mode_s	dsa_set_init_mode_s
K93		dsa_get_init_final_phase_s	dsa_set_init_final_phase_s
K94		dsa_get_init_time_s	dsa_set_init_time_s
K97		dsa_get_init_initial_phase_s	dsa_set_init_initial_phase_s
KF1	KPP	dsa_get_pl_proportional_gain_s	dsa_set_pl_proportional_gain_s
KF1 or KF61		dsa_get_pl_proportional_gain_ex_s	dsa_set_pl_proportional_gain_ex_s
KF191		dsa_force_control_get_default_force_range_s	dsa_force_control_set_default_force_range_s
KF2	KDP	dsa_get_pl_speed_feedback_gain_s	dsa_set_pl_speed_feedback_gain_s
KF2 or KF62		dsa_get_pl_speed_feedback_gain_ex_s	dsa_set_pl_speed_feedback_gain_ex_s
KF205	CAM	dsa_get_came_value_s	dsa_set_came_value_s
KF21	KAFFP	dsa_get_pl_acc_feedforward_gain_s	dsa_set_pl_acc_feedforward_gain_s
KF224		dsa_get_ctrl_gain_s	dsa_set_ctrl_gain_s
KF224:sidx		dsa_get_ctrl_gain_ex_s	dsa_set_ctrl_gain_ex_s
KF339:(pulse_-generator)		dsa_trigger_get_pulse_generator_period_coeff_s	dsa_trigger_set_pulse_generator_period_coeff_s
KF342:(pulse_-generator)		dsa_trigger_get_pulse_generator_delay_coeff_s	dsa_trigger_set_pulse_generator_delay_coeff_s
KF343:(pulse_-generator)		dsa_trigger_get_pulse_generator_pulse_width_co-eff_s	dsa_trigger_set_pulse_generator_-pulse_width_coeff_s
KF360		dsa_trigger_get_time_compensation_s	dsa_trigger_set_time_compensation_s
KF4	KIP	dsa_get_pl_integrator_gain_s	dsa_set_pl_integrator_gain_s
KF4 or KF64		dsa_get_pl_integrator_gain_ex_s	dsa_set_pl_integrator_gain_ex_s
KF44		dsa_get_homing_current_limit_s	dsa_set_homing_current_limit_s
KF5	KAWP	dsa_get_pl_anti_windup_gain_s	dsa_set_pl_anti_windup_gain_s
KF6		dsa_get_pl_integrator_limitation_s	dsa_set_pl_integrator_limitation_s
KF60	IPEAK	dsa_get_software_current_limit_s	dsa_set_software_current_limit_s
KF72		dsa_get_encoder_phase_1_factor_s	dsa_set_encoder_phase_1_factor_s
KF73		dsa_get_encoder_phase_2_factor_s	dsa_set_encoder_phase_2_factor_s
KF80	KPC	dsa_get_cl_proportional_gain_s	dsa_set_cl_proportional_gain_s
KF81	KIC	dsa_get_cl_integrator_gain_s	dsa_set_cl_integrator_gain_s
KF83		dsa_get_cl_current_limit_s	dsa_set_cl_current_limit_s
KF84		dsa_get_cl_i2t_current_limit_s	dsa_set_cl_i2t_current_limit_s
KF85		dsa_get_cl_i2t_time_limit_s	dsa_set_cl_i2t_time_limit_s
KF91		dsa_get_init_pulse_level_s	dsa_set_init_pulse_level_s
KF92		dsa_get_init_max_current_s	dsa_set_init_max_current_s
KL206	BRKDEC	dsa_get_brake_deceleration_s	dsa_set_brake_deceleration_s
KL210:depth		dsa_get_target_position_s	dsa_set_target_position_from_register_s
KL211	SPD	dsa_get_profile_velocity_s	dsa_set_profile_velocity_s
KL212	ACC	dsa_get_profile_acceleration_s	dsa_set_profile_acceleration_s
KL27		dsa_get_max_position_range_limit_s	dsa_set_max_position_range_limit_s
KL34	MINSL	dsa_get_min_soft_position_limit_s	dsa_set_min_soft_position_limit_s
KL35	MAXSL	dsa_get_max_soft_position_limit_s	dsa_set_max_soft_position_limit_s
KL41	HSPD	dsa_get_homing_zero_speed_s	dsa_set_homing_zero_speed_s
KL42	HACC	dsa_get_homing_acceleration_s	dsa_set_homing_acceleration_s
KL43		dsa_get_homing_following_limit_s	dsa_set_homing_following_limit_s
KL45	HOFFS	dsa_get_home_offset_s	dsa_set_home_offset_s
KL46		dsa_get_homing_fixed_mvt_s	dsa_set_homing_fixed_mvt_s
KL47		dsa_get_homing_switch_mvt_s	dsa_set_homing_switch_mvt_s
KL48		dsa_get_homing_index_mvt_s	dsa_set_homing_index_mvt_s
M10		dsa_get_velocity_demand_value_s	

Registers	Alias	EDI read function	EDI write function
M11		dsa_get_velocity_actual_value_s	
M14		dsa_get_acc_demand_value_s	
M140		dsa_get_drive_fuse_status_s	
M2		dsa_get_position_ctrl_error_s	
M2:sub_axis		dsa_get_position_ctrl_error_ex_s	
M25		dsa_get_cl_lkt_phase_1_s	
M25:1		dsa_get_cl_lkt_phase_2_s	
M25:2		dsa_get_cl_lkt_phase_3_s	
M282:what		dsa_get_parameters_version_s	
M3		dsa_get_position_max_error_s	
M345:(pulse_-generator)		dsa_trigger_get_pulse_generator_sent_pulse_number_s	
M346		dsa_trigger_get_treated_event_number_s	
M349		dsa_trigger_get_nb_free_event_s	
M350		dsa_trigger_get_first_free_event_index_s	
M351		dsa_trigger_get_treated_event_number_per_group_s	
M395		dsa_force_control_get_force_duration_s	
M40	SINENC	dsa_get_encoder_sine_signal_s	
M41	COSENC	dsa_get_encoder_cosine_signal_s	
M449 or M524		dsa_get_nb_available_slots_s	
M475 / M470		dsa_get_sequence_code_usage_s	
M476 / M471		dsa_get_sequence_data_usage_s	
M477 / M472		dsa_get_sequence_source_usage_s	
M48		dsa_get_encoder_hall_dig_signal_s	
M50	DIN	dsa_get_digital_input_s	
M52	FDIN	dsa_get_fast_digital_input_s	
M55	XDIN	dsa_get_x_digital_input_s	
M56:0		dsa_get_x_analog_input_1_s	
M56:1		dsa_get_x_analog_input_2_s	
M56:2		dsa_get_x_analog_input_3_s	
M56:3		dsa_get_x_analog_input_4_s	
M60	SD1	dsa_get_drive_status_1_s	
M61	SD2	dsa_get_drive_status_2_s	
M64	ERRCD	dsa_get_error_text	
M64	ERRCD	dsa_gantry_get_error_code	
M64	ERRCD	dsa_get_error_code_s	
M66	WARNCD	dsa_get_warning_code_s	
M66	WARNCD	dsa_get_warning_text	
M67	SEQNBP	dsa_debug_sequence_get_nb_breakpoints_s	
M77	SEQTHRBKD	dsa_debug_sequence_get_break_thread_nb_s	
M87		dsa_get_axis_number_s	
M90	CTEMP	dsa_get_drive_temperature_s	
M95		dsa_get_drive_display_s	
M96		dsa_get_drive_sequence_line_s	
MF20	RCUR1	dsa_get_cl_current_phase_1_s	
MF21	RCUR2	dsa_get_cl_current_phase_2_s	
MF22	RCUR3	dsa_get_cl_current_phase_3_s	
MF30	TIQ	dsa_get_cl_demand_value_s	
MF31	RIQ	dsa_get_cl_actual_value_s	
MF395		dsa_force_control_get_force_range_s	

Registers	Alias	EDI read function	EDI write function
MF51	AIN	dsa_get_analog_input_s	
MF67		dsa_get_cl_i2t_value_s	
ML434		dsa_get_min_user_soft_position_limit_s	
ML435		dsa_get_max_user_soft_position_limit_s	
ML6		dsa_get_position_demand_value_s	
ML6:sub_axis		dsa_get_position_demand_value_ex_s	
ML7		dsa_get_position_actual_value_s	
ML7:sub_axis		dsa_get_position_actual_value_ex_s	

15.2 Main programming changes when passing from EDI3.xx to EDI4.xx

15.2.1 All files including EDI header files must include new version of header files

EDI3.xx	EDI4.xx
#include<etne20.h>	#include <etne40.h>
#include<tra20.h>	#include <tra40.h>
#include<dsa30.h>	#include <dsa40.h>
#include<etb20.h>	#include <etb40.h>
#include<dmd20.h>	#include <dmd40.h>
#include<ekd20.h>	#include <ekd40.h>
#include<lib20.h>	#include <lib40.h>
#include<esc10.h>	#include<esc40.h>

15.2.2 The application must be linked with new version of library files

EDI3.xx	EDI4.xx
-L etne20c.lib	-L etne40c.lib
-L tra20c.lib	-L tra40c.lib
-L dsa30c.lib	-L dsa40c.lib
-L etb20c.lib	-L etb40c.lib
-L dmd20c.lib	-L dmd40c.lib
-L ekd20c.lib	-L ekd40c.lib
-L lib20c.lib	-L lib40c.lib
-L esd10c.lib	-L esd40c.lib
-L esc10c.lib	-L esc40c.lib (available for 32-bits application only) -L emp40c.lib (from EDI4.10A)

15.2.3 EDI obsolete functions

15.2.3.1 Register and functionality remove

EDI, especially DSA library, provides a huge amount of specific functions. EDI4.xx does not support old DSA, DSB and DSC devices anymore. Some specific functions were designed to access DSA/DSB/DSC specific registers. These functions are no more available:

Obsolete functions due to register or functionality remove
dsa_set_trace_mode_mvt_s/a
dsa_set_trace_mode_pos_s/a
dsa_set_trace_mode_dev_s/a
dsa_set_trace_mode_iso_s /a
dsa_set_trace_mode_immediate_s/a
dsa_trace_acquisition_s/a
dsa_ipol_reset_s/a
dsa_wait_window_user_channel_s/a

Obsolete functions due to register or functionality remove
dsa_wait_movement_user_channel_s/a
dsa_wait_time_user_channel_s/a
dsa_wait_position_user_channel_s/a
dsa_wait_sgn_register_greater_user_channel_s/a
dsa_wait_sgn_register_lower_user_channel_s/a
dsa_wait_bit_set_user_channel_s/a
dsa_wait_bit_clear_user_channel_s/a
dsa_get_rtm_mon
dsa_init_rtm_fct
dsa_start_rtm
dsa_stop_rtm
dsa_edit_sequence_s/a
dsa_exit_sequence_s/a
dsa_can_command_1_s/a
dsa_can_command_2_s/a
dsa_get_ebl_baudrate_s/a
dsa_get_drive_fuse_checking_s/a
dsa_get_motor_temp_checking_s/a
dsa_get_interrupt_mask_1_s/a
dsa_get_interrupt_mask_2_s/a
dsa_get_indirect_axis_number_s/a
dsa_get_indirect_register_sidx_s/a
dsa_get_daisy_chain_number_s/a
dsa_get_drive_mask_value_s/a
dsa_get_irq_drive_status_1_s/a
dsa_get_irq_drive_status_2_s/a
dsa_get_ack_drive_status_1_s/a
dsa_get_ack_drive_status_2_s/a
dsa_get_irq_pending_axis_mask_s/a
dsa_get_encoder_phase_3_factor_s/a
dsa_get_encoder_phase_3_offset_s/a
dsa_get_encoder_index_signal_s/a
dsa_get_encoder_hall_1_signal_s/a
dsa_get_encoder_hall_2_signal_s/a
dsa_get_encoder_hall_3_signal_s/a
dsa_get_apr_input_filter_s/a
dsa_get_cl_regen_mode_s/a
dsa_get_pdr_step_value_s/a
dsa_get_ctrl_shift_factor_s/a
dsa_get_ref_demand_value_s/a
dsa_get_drive_control_mask_s/a
dsa_get_cl_output_filter_s/a
dsa_get_cl_input_filter_s/a
dsa_get_cl_phase_advance_factor_s/a
dsa_get_cl_phase_advance_shift_s/a
dsa_get_pl_speed_feedfwd_gain_s/a
dsa_get_pl_force_feedback_gain_1_s/a
dsa_get_pl_force_feedback_gain_2_s/a
dsa_get_pl_output_filter_s/a
dsa_get_pl_speed_filter_s/a
dsa_get_ttl_special_filter_s/a
dsa_get_init_current_rate_s/a
dsa_get_init_phase_rate_s/a
dsa_get_end_velocity_s/a
dsa_get_profile_deceleration_s/a
dsa_get_min_position_range_limit_s/a
dsa_get_max_profile_velocity_s/a
dsa_get_max_acceleration_s/a
dsa_get_acc_actual_value_s/a

Obsolete functions due to register or functionality remove
dsa_get_io_error_event_mask_s/a
dsa_get_syncro_input_mask_s/a
dsa_get_syncro_input_value_s/a
dsa_get_syncro_output_mask_s/a
dsa_get_syncro_output_value_s/a
dsa_get_x_analog_gain_s/a
dsa_get_x_analog_offset_s/a
dsa_get_can_feedback_1_s/a
dsa_get_can_feedback_2_s/a
dsa_get_mon_dest_index_s/a
dsa_get_mon_gain_s/a
dsa_get_mon_offset_s/a
dsa_get_trigger_map_offset_s/a
dsa_get_trigger_map_size_s/a
dsa_get_trigger_io_mask_s/a
dsa_get_trigger_irq_mask_s/a
dsa_get_realtime_enabled_global_s/a
dsa_get_realtime_valid_mask_s/a
dsa_get_realtime_enabled_mask_s/a
dsa_get_realtime_pending_mask_s/a
dsa_set_ebl_baudrate_s/a
dsa_set_drive_fuse_checking_s/a
dsa_set_motor_temp_checking_s/a
dsa_set_interrupt_mask_1_s/a
dsa_set_interrupt_mask_2_s/a
dsa_set_indirect_axis_number_s/a
dsa_set_indirect_register_sidx_s/a
dsa_set_encoder_phase_3_factor_s/a
dsa_set_encoder_phase_3_offset_s/a
dsa_set_apr_input_filter_s/a
dsa_set_cl_regen_mode_s/a
dsa_set_pdr_step_value_s/a
dsa_set_ctrl_shift_factor_s/a
dsa_set_cl_output_filter_s/a
dsa_set_cl_input_filter_s/a
dsa_set_cl_phase_advance_factor_s/a
dsa_set_cl_phase_advance_shift_s/a
dsa_set_pl_speed_feedfwd_gain_s/a
dsa_set_pl_force_feedback_gain_1_s/a
dsa_set_pl_force_feedback_gain_2_s/a
dsa_set_pl_output_filter_s/a
dsa_set_pl_speed_filter_s/a
dsa_set_ttl_special_filter_s/a
dsa_set_init_current_rate_s/a
dsa_set_init_phase_rate_s/a
dsa_set_end_velocity_s/a
dsa_set_profile_deceleration_s/a
dsa_set_min_position_range_limit_s/a
dsa_set_max_profile_velocity_s/a
dsa_set_max_acceleration_s/a
dsa_set_io_error_event_mask_s/a
dsa_set_syncro_input_mask_s/a
dsa_set_syncro_input_value_s/a
dsa_set_syncro_output_mask_s/a
dsa_set_syncro_output_value_s/a
dsa_set_x_analog_gain_s/a
dsa_set_x_analog_offset_s/a
dsa_set_mon_dest_index_s/a
dsa_set_mon_gain_s/a

Obsolete functions due to register or functionality remove
dsa_set_mon_offset_s/a
dsa_set_trigger_map_offset_s/a
dsa_set_trigger_map_size_s/a
dsa_set_trigger_io_mask_s/a
dsa_set_trigger_irq_mask_s/a
dsa_set_realtime_enabled_global_s/a
dsa_set_realtime_valid_mask_s/a
dsa_set_realtime_pending_mask_s/a
etb_get_baudrate
etb_multi_send
etb_start_rtm
etb_stop_rtm
etb_init_rtm_fct
etb_get_rtm_mon
etb_link_error
etb_irq_watchdog
etb_get_bus_counters
etb_add_rt_handler
etb_remove_rt_handler
etb_auto_number
etb_activate_download
etb_start_download_file
etb_start_upload_file
etb_etcom_multi_send
etb_etcom_start_download_file
etb_etcom_start_upload_file

15.2.3.2 Axis addressing

DSC and AccurET families differ concerning axis addressing:
- DSC family allows 32 axes, where DSMAX has number 31.
- AccurET family allows 64 axes, where UltimET has number 63.

EDI functions with parameters allowing axis addressing have been doubled in EDI3.xx (for e.g. `dsa_get_etb_axis` was dedicated to DSC family and returned an axis number 0 and 31, while `dsa_etcom_get_etb_axis` was dedicated to AccurET family and returned an axis number 0 and 63). As EDI4.xx does not support DSC any more, the DSC dedicated functions addressing axis have been removed:

Obsolete functions due to DSC family axis addressing
dsa_open_e
dsa_open_ef
dsa_get_etb_axis
dsa_create_auto_e
etb_get_baudrate
etb_multi_send
etb_get_drv_present
etb_get_drv_status
etb_get_drv_info
etb_get_ext_info
etb_diag
etb_sdiag
etb_fdiag
etb_putm
etb_putr
etb_getm
etb_getr
etb_start_download
etb_start_upload
etb_download_firmware

Obsolete functions due to DSC family axis addressing
tra_upload_register_stream_e
tra_download_register_stream_e
tra_download_register_stream_e2
tra_upload_limited_register_stream_e
tra_send_direct_stream_e
tra_get_axis_mask_e
tra_set_axis_mask_e
tra_set_preference_axis_mask
tra_get_preference_axis_mask

15.2.3.3 Record accessing

DSC and AccurET families differ concerning the record used on the communication protocol:
- DSC family uses ETBREC fixed size record.
- AccurET family uses ETCOM variable length record.

As EDI3.xx supports both DSC and ETEL products, some functions allowing management of records have been implemented. The functions for the old ETBREC record have been removed:

Obsolete functions due to DSC-family ETBREC record
tra_translate_cmd_to_ascii
tra_translate_cmd_to_ascii_ex
tra_translate_cmd_from_ascii
tra_translate_cmd_from_ascii_ex
tra_translate_rqs_to_ascii
tra_translate_rqs_to_ascii_ex
tra_get_iso_converter
tra_set_iso_converter
etb_etcom_to_recs
etb_recs_to_etcom

15.2.3.4 NON ANSI functions

Some non ANSI functions have been removed. These are the functions returning a whole structure content. These are especially the functions allowing the initialization of a structure:

Obsolete non ANSI functions
dsa_init_status
dsa_init_info
dsa_init_x_info
dsa_init_vector
dsa_init_vector_typ
dsa_init_rtm
etb_init_drv_info
etb_init_timeouts
etb_init_ext_info
etb_init_drv_status
etb_init_rec_param
etb_init_counters
etb_init_bus_status
etb_init_svr_info
etb_activate_status
etb_init_master_info

The call to these functions can be replaced by:

Example:

```
DSA_STATUS status = dsa_init_status();
```

replaced by

```
DSA_STATUS status = {sizeof(DSA_STATUS)};
```

15.2.3.5 Functions managing special ETEL devices

A special ETEL device called 'gp_module' was developed. This device was a special 'TEB' device. All functions accessing this device have been removed:

Obsolete 'GP_MODULE' special device functions
dsa_create_gp_module
dsa_create_gp_module_group
dsa_is_valid_gp_module
dsa_is_valid_gp_module_group
dsa_is_valid_gp_module_base

15.2.3.6 Special functions accessing 'DSMAX'

Some functions accessing DSMAX must be used in EDI3.xx to access UltimET as well. These functions have been renamed into '...MASTER...', providing a more generic name. To allow portability between EDI3.xx and EDI4.xx, the old functions name still exists, but will be removed in the future:

Obsolete 'DSMAX' functions	
dsa_create_dsmax	dsa_create_master
dsa_create_ dsmax _group	dsa_create_ master _group
dsa_is_valid_ dsmax	dsa_is_valid_master
dsa_is_valid_dsmax _group	dsa_is_valid_master _group
dsa_is_valid_dsmax_base	dsa_is_valid_master_base
dsa_set_dsmax	dsa_set_master
dsa_get_dsmax	dsa_get_master

Obsolete 'DSMAX' objects	
DSA_DSMAX	DSA_MASTER
DSA_DSMAX_GROUP	DSA_MASTER_GROUP
DSA_DSMAX_BASE	DSA_MASTER_BASE

15.2.3.7 Function allowing old ETEL sequence translation/download/upload

The ETEL Sequences on DSC family were interpreted by the controller. These Sequences were saved into S registers of the controller. The functions, allowing translation, download and upload of such Sequences into S registers have been removed:

Obsolete ETEL sequence translation/download/upload
tra_is_valid_sequence_traductor
tra_create_sequence_traductor_o
tra_create_sequence_traductor_o2
tra_clear_sequence_drive_map
tra_setup_sequence_drive_map
tra_etcom_setup_sequence_drive_map
tra_is_valid_sequence_traductor_e
tra_create_sequence_traductor_e
tra_download_sequence_stream_e
tra_upload_sequence_stream_e
tra_get_sequence_line_e

15.3 Diagnostic (DSA_EXT_DIAG()) output description

DSA_DIAG() outputs a mixture of valuable information meant in part for the developer of the application, and in part for the ETEL DLL customer support should that be necessary. The output of DSA_EXT_DIAG is the standard output. If the user application does not redirect its standard output or if the standard output is not visible, it is advised to use DSA_EXT_SDIAG or even DSA_EXT_FDIAG to store the error message.

The layout of the output comes in three parts:

- The first line is very important for the application developer, because it provides a report on the state of the application: communication status and drive status. With this line and knowledge of the application and the system, the user should be able to identify the source of the problem in a large majority of the cases.
- The following lines constitutes the calling stack trace and is primarily meant for ETEL Support. It gives precise information on the execution flow (file name, function called and line number) that led to the error. It is for this piece of information that it is strongly recommended to systematically call DSA_EXT_DIAG() after each operation used from the EDI package. **This information will be requested by ETEL's Support team anytime a problem is reported.**
- The last line is a summarizing error message

15.3.1 Example 1

Programming example:

```
DSA_DRIVE *drv1 = NULL;
int err = 0;
if (err = dsa_create_drive(drv1)) { //Should be &drv1
     DSA_EXT_DIAG(err, drv1);
     goto _error;
}
```

Output:

```
.\test1.c (11045) : ERROR -322 : bad parameter (EDI package 0x4238080)
stacktrace:
      X:\lib-edi\sw_dsa\v40\c\dsaopn.c (dsa_create_drive:645)
      ERROR -322 : bad parameter : NULL DSA_DRIVE pointer on pointer
```

In this case, the user passes the pointer to the object instead of the address. By calling dsa_create_drive function, EDI checks if the pointer is correct. If it is not the case, it returns a DSA_EBADPARAM error. DSA_EXT_DIAG is called by passing the pointer as parameter. DSA_EXT_DIAG prints out that the pointer is not valid.

15.3.2 Example 2

Programming example:

```
DSA_DRIVE *drv1 = NULL;
int err = 0;

if (err = dsa_create_drive(&drv1)) {
     DSA_EXT_DIAG(err, NULL);
     goto _error;
}
if (err = dsa_open_u(drv1, "etb:ULTIMET:2")) {
     DSA_EXT_DIAG(err, drv1);
     goto _error;
}
```

Output:

```
.\test1.c (11048) : ERROR -320 : no drive response (EDI package 0x4238080)
```

```
on unknown command 0 (record 0x0, timeout 0 ms)sent on axisMask 0x0
status device 2 (status 0xc)
     POWER ON: 0
     PRESENT: 0
     MOVING: 0
     IN WINDOW: 0
     SEQUENCE ON: 0
     ERROR: 0
     TRACE ON: 0
     SEQUENCE THREAD ON: 0
     WARNING: 0
     SAVE_POS: 0
     BREAKPOINT: 0
     USER-BITS: 0x0
stacktrace:
     X:\lib-edi\sw_dsa\v40\c\dsaopn.c (dsa_open_u:1241)
     X:\lib-edi\sw_dsa\v40\c\dsaopn.c (dsa_reset:1529)
     ERROR -320 : no drive response : Device not present on bus
```

In this case, the user tries to open an axis not present. The call to `dsa_open_u` function returns an error. `DSA_EXT_DIAG` prints out that axis 2 does not respond. The user can also see this by analyzing the printed out drive status (0x0 0x0).

15.3.3 Example 3

Programming example:

```
DSA_DRIVE *drv1 = NULL;
DSA_DRIVE *drv2 = NULL;
DSA_DRIVE_GROUP *grp = NULL;
int err = 0;

if (err = dsa_create_drive(&drv1)) {
   DSA_EXT_DIAG(err, NULL);
   goto _error;
}
if (err = dsa_create_drive(&drv2)) {
   DSA_EXT_DIAG(err, drv2);
   goto _error;
}
if (err = dsa_create_drive_group(&grp, 2)) {
   DSA_EXT_DIAG(err, grp);
   goto _error;
}
if (err = dsa_set_group_item(grp, 0, drv1)) {
   DSA_EXT_DIAG(err, drv1);
   goto _error;
}
if (err = dsa_set_group_item(grp, 1, drv2)) {
   DSA_EXT_DIAG(err, drv2);
   goto _error;
}
if (err = dsa_open_u(drv1, "etb:ULTIMET:0")) {
   DSA_EXT_DIAG(err, drv1);
   goto _error;
}
if (err = dsa_open_u(drv2, "etb:ULTIMET:1")) {
   DSA_EXT_DIAG(err, drv2);
   goto _error;
```

```
  }
  if (err = dsa_wait_time_s(grp, 5.0, 10)) {
    DSA_EXT_DIAG(err, grp);
    goto _error;
  }
```

Output:

```
.\test1.c (11067) : ERROR -310 : timeout error (EDI package 0x4238080)on command WTT
(record 0x20, timeout 10 ms) sent on axisMask 0x3
status device 0 (status 0xc)
  POWER ON: 0
  PRESENT: 1
  MOVING: 0
  IN WINDOW: 1
  SEQUENCE ON: 0
  ERROR: 1 (device-error 56: Time-out error in TransnET communication)
  TRACE ON: 0
  SEQUENCE THREAD ON: 0
  WARNING: 0
  SAVE_POS: 0
  BREAKPOINT: 0
  USER-BITS: 0x0
status device 1 (status 0xc)
  POWER ON: 0
  PRESENT: 1
  MOVING: 0
  IN WINDOW: 1
  SEQUENCE ON: 0
  ERROR: 1 (device-error 56: Time-out error in TransnET communication)
  TRACE ON: 0
  SEQUENCE THREAD ON: 0
  WARNING: 0
  SAVE_POS: 0
  BREAKPOINT: 0
  USER-BITS: 0x0
stacktrace:
  X:\lib-edi\sw_dsa\v40\c\dsawai.c (dsa_wait_time_s:174)
  X:\lib-edi\sw_dsa\v40\c\dsasyn.c (_dsa_sync_cmd_2c:902)
  X:\lib-edi\sw_dsa\v40\c\dsasyn.c (_dsa_sync_cmd_ncn:600)
  X:\lib-edi\sw_dsa\v40\c\dsasyn.c (_dsa_sync_cmd_ncx:441)
  X:\lib-edi\sw_dsa\v40\c\dsasyn.c (_dsa_sync_cmd_g1cxv:1022)
  X:\lib-edi\sw_dsa\v40\c\dsatrn.c (dsa_commit_sync_trans:205)
  X:\lib-edi\sw_dsa\v40\c\dsamsg.c (_dsa_wait_sync_event:922)
  ERROR -310 : timeout error : Event waiting timeout
  AxisMask 0x3
  Rec 0x20, Cmd 0xa, Timeout 10 [ms]
```

In this case, the user calls the WTT function with an EDI timeout of 10 ms. This follows to a `DSA_ETIMEOUT` error. `DSA_EXT_DIAG` is called passing the device group as parameter. `DSA_EXT_DIAG` prints out the error and the status of each drive present in the group.

15.3.4 Example 4

```
  DSA_DRIVE *drv1 = NULL;
  DSA_DRIVE *drv2 = NULL;
  int err = 0;
```

```
  if (err = dsa_create_drive(&drv1)) {
    DSA_EXT_DIAG(err, NULL);
    goto _error;
  }
  if (err = dsa_create_drive(&drv2)) {
    DSA_EXT_DIAG(err, drv2);
    goto _error;
  }
  if (err = dsa_open_u(drv1, "etb:ULTIMET:0")) {
    DSA_EXT_DIAG(err, drv1);
    goto _error;
  }
  if (err = dsa_open_u(drv2, "etb:ULTIMET:1")) {
    DSA_EXT_DIAG(err, drv2);
    goto _error;
  }
  if (err = dsa_power_on_s(drv1, 1000)) {
    DSA_EXT_DIAG(err, drv1);
    goto _error;
  }
  if (err = dsa_homing_start_s(drv1, 1000)) {
    DSA_EXT_DIAG(err, drv1);
    goto _error;
  }
```

Output:

```
.\test1.c (11056) : ERROR -311 : drive in error (EDI package 0x4238080)on command PWR
(record 0x20, timeout 1000 ms) sent on axisMask 0x1
status device 0 (status 0xc)
   POWER ON: 0
   PRESENT: 1
   MOVING: 0
   IN WINDOW: 1
   SEQUENCE ON: 0
   ERROR: 1 (device-error 56: Time-out error in TransnET communication)
   TRACE ON: 0
   SEQUENCE THREAD ON: 0
   WARNING: 0
   SAVE_POS: 0
   BREAKPOINT: 0
   USER-BITS: 0x0
stacktrace:
   X:\lib-edi\sw_dsa\v40\c\dsafct.c (dsa_power_on_s:405)
   X:\lib-edi\sw_dsa\v40\c\dsasyn.c (_dsa_sync_cmd_2c:902)
   X:\lib-edi\sw_dsa\v40\c\dsasyn.c (_dsa_sync_cmd_ncn:600)
   X:\lib-edi\sw_dsa\v40\c\dsasyn.c (_dsa_sync_cmd_ncx:561)
   X:\lib-edi\sw_dsa\v40\c\dsamsg.c (_dsa_wait_sync_event:918)
   ERROR -311 : drive in error : Event received with message error
   AxisMask 0x1
   Rec 0x20, Cmd 0x7c, Timeout 1000 [ms]
```

In this case, the user calls `dsa_homing_start` on a drive which is not powered on. This returns a `DSA_EDRVERR` error. `DSA_EXT_DIAG` prints out that the axis 0 cannot be homed because it is not powered on.

15.4 Description of the other examples in C

Example1: acquisition\acquisition.c

```
/*
* This simple demo program shows how to make acquisition, using ETEL EDI library set. This example will do
* an acquisition of one trace on two AccurET controllers connected to an UltimET.
*/

/*
* To run this demo without modification, you must have:
* - 2 linear motor and an AccurET properly configured.
* - an UltimET plugged in the PC
* - AccurET and UltimET connected through TransnET
*/
```

Example2: compiler\compiler.c

```
/*
* This simple demo program shows how to:
* - compile and download a Sequence into an AccurET.
* - upload a Sequence from an AccurET
*/

/*
* To run this demo without modification, you don't need to have a motor connected. You must have one or many
* AccurET(s) connected through USB or UltimET. The example allows to choose between several
* configurations
*/
```

Example3: compiler\offline_compiler.c

```
/*
* This simple demo program shows how to:
* - compile a sequence using an off-line compiler.
* - download the compiled Sequence into a device
*/

/*
* To run this demo without modification, you don't need to have a motor connected. You must have one or any
* AccurET(s) connected through USB or UltimET. The example allows to choose between several
* configurations
*/
```

Example4: generic\sample_1.c

```
/*
* This simple demo program shows how to make basic drive operations (power on, indexation, movement) on
* a single drive, using ETEL EDI library set.
* The program will send a power on and an indexation command, then move the motor near the two limits of
* the available range, go to the zero position again and power off the drive.
*/

/*
* To run this demo without modification, you must have:
* - a linear motor and an AccurET controller properly configured.
* - a USB connection between the drive and the PC.
* - the drive must have the axis number 0.
* - KL45 must be set to insure that position 0 is inside the valid range.
* - KL34 and KL35 must be set properly.
* If KL34 and KL35 aren't set properly, you can modify this code to set pos_min and pos_max manually.
*/
```

Example5: generic\sample_2.c

```
/* This second sample program extends sample_1.c and shows how to use the library to control multiple axes
* in once. It will move a x/y table to random positions.
*/

/*
* To run this demo without modification, you must have:
* - two linear motors with one AccurET controller configured.
* - a USB connection between the drive and the PC.
* - the drives must have the axes number 0 and 1.
* - KL45 must be set to insure that position 0 is inside the valid range.
* - KL34 and KL35 must be set properly.
*/
```

Example6: generic\sample_3.c

```
/*
* This third sample program extends sample_2.c and shows how to use multithreading with ETEL's DLLs. In
* this program, we will add a thread to monitor the current position of the drive and a thread for emergency stop.
* The monitoring thread will show the current position of both axes every 100ms, and the second thread will
* wait indefinitely for a command from the user: the space key will immediately stop the sequence, which will
* run indefinitely otherwise.
*/

/*
* To run this demo without modification, you must have:
* - two linear motors with one AccurET controller configured.
* - a USB connection between the drive and the PC.
* - the drives must have the axes number 0 and 1.
* - KL45 must be set to insure that position 0 is inside the valid range.
* - KL34 and KL35 must be set properly.
*/
```

Example7: generic\sample_4.c

```
/*
C example for upload / download of parameters in an UltimET.
The functions defined in this example are also valid for AccurET drives.
*/
```

Example8: generic\sample_5.c

```
/*
C example for download firmware into a DRIVE or an UltimET.
*/
```

Example9: io\io1.c

```
/*
* This sample program show how to set and get I/Os from UltimET.
*/

/*
* To run this demo without modification, you must have:
* - an UltimET PCI board plugged into your PC or
* - an UltimET TCP/IP connected to the network
*/
```

Example10: mapping\mapping1.c

```
/*
* This sample program shows how to:
*    -    download mapping file into a group of drives
*    -    upload data stored in a group of device into a mapping file
*    -    activate mapping of a group of drive
```

```
*    -    deactivate mapping of a group of drive
*    -    check mapping activity of a group of drive
*/

/*
* To run this demo without modification, you must have:
* - a PCI-UltimET board plugged into your PC
* - one or many AccurET(s) connected to UltimET through TransnET
* If you want to run this example without UltimET, you must change the URL It is advised to download first
* mapping files provided with the example before uploading data from drives. There are 2 mapping files
* provided with the example:
* - map2D.txt which is a correct mapping file downloadable into 2 AccurET number 0 and 1.
* - map3D.txt which is a correct mapping file downloadable into 3 AccurET number 0, 1 and 2.
*/
```

Example11: mapping\scaling1.c

```
/*
* This sample program shows how to
*    -    download scaling file into a drive
*    -    activate scaling of a drive
*    -    deactivate scaling of a drive
*    -    check scaling activity of a drive
*/
/*
* To run this demo without modification, you must have:
* - an AccurET connected to your PC, either by USB, TCP/IP or through an UltimET
* Just change the URL to fit the connection
* There is 1 scaling file provided with the example:
* - scaling.txt which is a correct scaling file downloadable into an AccurET number 0.
*/
```

Example12: rtv\rtv1.c

```
/*
* This sample program shows how to read real-time value on TransnET without irq synchronization, this means
* that the software will configure UltimET to put:
*        M50: Digital input
*        ML7: real position real-time value
*        MF31: Real force Iq measured of the AccurET into Real time slot on TransnET.
* The example will then read these value asynchronously.
*/
/*
* To run this demo without modification, you must have:
* - a PCI-UltimET board plugged into your PC
* - an AccurET device 0 connected to UltimET through TransnET
* - Eventually a connected and set motor
*/
```

Example13: rtv\rtv2.c

```
/*
* This sample program shows how to read real-time value on TransnET with irq synchronization, this means
* that the software will configure UltimET to put:
*        M50: Digital input
*        ML7: real position real-time value
*        MF31: Real force Iq measured of the AccurET into Real-time slot on TransnET.
* The example will then read these value synchronously when DSA_RTV_HANDLER will be called
*/
/*
* To run this demo without modification, you must have:
* - a PCI-UltimET board plugged into your PC
* - an AccurET device 0 connected to UltimET through TransnET
* - Eventually a connected and set motor
*/
```

Example14: rtv\rtv3.c

```
/*
* This sample program shows how to setup slave-to-slave communication using RTV.
* - Configure Axis 0 to put ML1 value into a slot
* - Configure Axis 1 to read slot value into ML450
* - Move Drive 0 and monitor ML450 of drive 1 during move
* This is an example. Slave to slave communication can be made without RTV if the two axes are part of the
* same AccurET
*/

/*
* To run this demo without modification, you must have:
* - a PCI-UltimET board plugged into your PC
* - an AccurET axis 0 connected to UltimET through TransnET with a connected and set motor
* - an AccurET axis 1 connected to UltimET through TransnET
*/
```

Example15: stream\stream.c

```
/*
* This sample program shows how to make "stream" transfer between PC and device. Stream transfer allows
* to write or read a large amount of data into/from the device. This type of transfer is mainly used when
* uploading traces or registers or when downloading firmwares.
* This sample allows the user to choose the communication bus and the connected devices. No tests are done
* concerning the validity of the registers' typ. For example, if is not possible to write M registers, but this
* example does no validity check
*/
```

Example16: ultimet\ultimet_1.c

```
/*
* This simple demo program shows how to make basic interpolated movements on a X-Y axes system using
* ETEL's EDI libraries and the UltimET motion controller. The program will switch the motors on, send an
* indexation command, make a simple movement, switch to the interpolation mode and interpolate a G-code
* movement.
*/

/*
* To run this demo without modification, you must have:
* - two linear motors with one AccurET controllers configured.
* - one UltimET Light PC, configured like this
* - a TransnET connection between the drives and the UltimET.
* - the drives must have the axis number 0 and 1.
* - KL45 must be set to insure that position 0 is inside the valid range.
* - KL34 and KL35 must be set properly.
*/
```

Example17: ultimet\ultimet_2.c

```
 /*
* This is a simple program that use the UltimET motion controller to perform an interpolated movement on a
* X-Y axes system with PVT commands. PVT means Position - Velocity - Time. A movement is defined by the
* position of the destination point (in x,y,z and theta coordinates), the velocity at the destination point, and the
* time to do the displacement.
*/

/*
* To run this demo without modification, you must have:
* - two linear motors with one AccurET controllers configured.
* - one UltimET Light PCI, configured like this
* - a TransnET connection between the drives and the UltimET.
* - the drives must have the axis number 0 and 1.
* - KL45 must be set to insure that position 0 is inside the valid range.
```

```
* - KL34 and KL35 must be set properly.
*/
```

Example18: ultimet\ultimet_3.c

```
/*
* This is a simple program that illustrates the working principle of the command 126.
* The aim of this function is to set in one block several values into X, XL, XF, XD or L registers. 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. It sets from a specified index at a fixed depth the values in the specified type of register.
* This command has to be sent by calling dsa_execute_command_x_s(). Please see the EDI HTML
* documentation for further explanations about this function.
* The number of parameters to send has to be equal to the number of effective values plus one. This additional
* parameter has to be first set to specify the type of register, the depth and the index used by the command.
*/

/*
* To run this demo without modification, you must have:
* - one UltimET Light PCI or an UltimET TCP/IP
*/
```

Example 19: etne/etne_sample1.c

```
/*
* This software is an example showing the way to implement the following things:
* 1) The way to open several connection on an UltimET PCI using ETNE server
* 2) How to use the new DSA_EXT_DIAG for extended diagnostic in case of error
*
* Remark: The EDI package must be greater or equal to 4.00A!!!
*/
```

Example 20: trigger/trigger_sample1.c

```
/*
* This example shows how to set a trigger using dsa specific functions.
*/
```

Example 21: umeg\umeg.c (User's manual example)

```
/*
*This is the example that is referred to in the “EDI User's Manual”.
*
* This examples aims at using the library to control two axes concurrently via an UltimET.
* The UltimET is an UltimET-PCI and the controllers are one AccurET 400.
* If an UltimET Light TCP/IP or UltimET Advanced is used, only the URL used when the communication is
* opened must be changed. The remainder of the example will run.
* A first set of actions aims at getting the drives in an operational state which entails:
* - establishing the communication with the UltimET and the drives connected to it
* - powering up the drives
* - performing the homing procedure on each drive.
*
* The next set deals with making movements on each axis by first describing how to set the limits and precision
* windows, and then actually specifying and starting the movement. The movements will be controlled by the
* UltimET which will execute them in interpolated mode. In a more elaborate step, one thread will be created
* to monitor the current position of the motors. The monitoring thread will show the current position of each
* motor every 100ms. Another will loop indefinitely waiting for a user input: the space bar will immediately stop
* the movement and set one of the controller’s digital outputs.
*
* Finally, it will be shown how to power off the drives.
*
* The last portion of the program covers error handling.
*
```

```
* To run this example without modification, you must have:
* - two linear motors and two ETEL position loop controllers set and tuned accordingly.
* - an UltimET Light PCI on a PCI slot of your PC.
* - a TransnET connection between the UltimET and the drives.
* - the drives must have the axis numbers 0 and 1.
* - the software position limits (KL34 and KL35) must have been set when setting the controllers otherwise
* these will be defaulted to zero and nothing will move.
*/
```

15.5 Deprecated functions

The table below lists the EDI deprecated C functions. Deprecated functions remain in the EDI package, they are functional, but not further improved.

ETEL S.A. strongly recommends against using these functions for the development of new applications, as they might be removed at the earliest in the next major or minor release.

15.5.1 C interface

DLL	Deprecated function	Recommended alternative function
DSA	int dsa_change_setpoint_s/a	int dsa_new_setpoint_s/a
DSA	int dsa_start_profiled_movement_s/a	int dsa_start_profiled_movement_ex_s/a
DSA	Int dsa_start_relative_profiled_movement_s/a	int dsa_start_relative_profiled_movement_ex_s/a
DSA	int dsa_set_profiled_movement_s/a	int dsa_set_profiled_movement_ex_s/a
DSA	int dsa_set_register_s/a	int dsa_set_register_int32_s/a
DSA	int dsa_get_register_s/a	int dsa_get_register_int32_s/a
DSA	int dsa_set_array_s/a	int dsa_set_array_int32_s/a
DSA	int dsa_get_array_s/a	int dsa_get_array_int32_s/a
DSA	int dsa_create_dsmax	int dsa_create_master
DSA	int dsa_create_dsmax_group	int dsa_create_master_group
DSA	int dsa_set_dsmax	int dsa_set_master
DSA	int dsa_get_dsmax	int dsa_get_master
DSA	ebool dsa_is_valid_dsmax	ebool dsa_is_valid_master
DSA	ebool dsa_is_valid_dsmax_group	ebool dsa_is_valid_master_group
DSA	ebool dsa_is_valid_dsmax_base	ebool dsa_is_valid_master_base
DSA	int dsa_set_gate	N/A
DSA	int dsa_get_new_gate	N/A
DSA	int dsa_clear_gate	N/A
DSA	int dsa_convert_to_iso	int dsa_convert_int32_to_iso
DSA	int dsa_convert_from_iso	int dsa_convert_int32_from_iso
DSA	int dsa_diag	int dsa_ext_diag
DSA	int dsa_sdiag	int dsa_ext_sdiag
DSA	int dsa_fdiag	int dsa_ext_fdiag
DSA	int dsa_acquisition_config_trigger	int dsa_acquisition_config_immediate_trigger (+ *other specific trigger functions*)
DSA	int dsa_acquisition_config_frequency	int dsa_acquisition_config_with_nb_points_and_total_time
DSA	int dsa_download_compiled_sequence_file	int dsa_download_compiled_sequence_file_ex
DSA	int dsa_quick_register_request_s/a	int dsa_quick_register_int32_request_s/a
DSA	int dsa_reset_error_s/a	int dsa_reset_error_ex_s/a
DSA	int dsa_reset_error_with_check_s/a	int dsa_reset_error_with_check_ex_s/a
DSA	int dsa_execute_sequence_s/a	int dsa_execute_sequence_in_thread_s/a
DSA	int dsa_stop_sequence_s/a	int dsa_stop_sequence_in_thread_s/a
DSA	int dsa_trigger_enable_s/a	int dsa_trigger_enable_ex_s/a
DSA	int dsa_trigger2d_enable_s/a	int dsa_trigger2d_enable_ex_s/a

DLL	Deprecated function	Recommended alternative function
DSA	int dsa_set_pl_speed_feedback_gain_s/a	int dsa_set_pl_speed_feedback_gain_ex_s/a
DSA	int dsa_get_pl_speed_feedback_gain_s/a	int dsa_get_pl_speed_feedback_gain_ex_s/a
DSA	int dsa_set_pl_proportional_gain_s/a	int dsa_set_pl_proportional_gain_ex_s/a
DSA	int dsa_get_pl_proportional_gain_s/a	int dsa_get_pl_proportional_gain_ex_s/a
DSA	int dsa_set_pl_integrator_gain_s/a	int dsa_set_pl_integrator_gain_ex_s/a
DSA	int dsa_get_pl_integrator_gain_s/a	int dsa_get_pl_integrator_gain_ex_s/a
DSA	int dsa_set_ctrl_source_index_s/a	int dsa_set_ctrl_source_index_ex_s/a
DSA	int dsa_get_ctrl_source_index_s/a	int dsa_get_ctrl_source_index_ex_s/a
DSA	int dsa_set_ctrl_source_type_s/a	int dsa_set_ctrl_source_type_ex_s/a
DSA	int dsa_get_ctrl_source_type_s/a	int dsa_get_ctrl_source_type_ex_s/a
DSA	int dsa_set_ctrl_offset_s/a	int dsa_set_ctrl_offset_ex_s/a
DSA	int dsa_get_ctrl_offset_s/a	int dsa_get_ctrl_offset_ex_s/a
DSA	int dsa_set_ctrl_gain_s/a	int dsa_set_ctrl_gain_ex_s/a
DSA	int dsa_get_ctrl_gain_s/a	int dsa_get_ctrl_gain_ex_s/a
DSA	int dsa_get_position_ctrl_error_s/a	int dsa_get_position_ctrl_error_ex_s/a
DSA	int dsa_get_position_demand_value_s/a	int dsa_get_position_demand_value_ex_s/a
DSA	int dsa_get_position_actual_value_s/a	int dsa_get_position_actual_value_ex_s/a
DSA	int dsa_force_control_set_s/a	int dsa_force_control_set_ex4_s/a
DSA	int dsa_force_control_set_ex1_s/a	int dsa_force_control_set_ex4_s/a
DSA	int dsa_force_control_set_ex2_s/a	int dsa_force_control_set_ex4_s/a
DSA	int dsa_force_control_set_ex3_s/a	int dsa_force_control_set_ex4_s/a
DSA	int dsa_force_control_reset_s/a	int dsa_force_control_reset_ex3_s/a
DSA	int dsa_force_control_reset_ex1_s/a	int dsa_force_control_reset_ex3_s/a
DSA	int dsa_force_control_reset_ex2_s/a	int dsa_force_control_reset_ex3_s/a
DSA	int dsa_force_control_new_s/a	int dsa_force_control_new_ex2_s/a
DSA	int dsa_force_control_new_ex1_s/a	int dsa_force_control_new_ex2_s/a
DSA	int dsa_force_control_set_immediate_s/a	int dsa_force_control_set_immediate_ex2_s/a
DSA	int dsa_force_control_set_immediate_ex1_s/a	int dsa_force_control_set_immediate_ex2_s/a
DSA	int dsa_start_upload_memory_s	N/A
DSA	int dsa_start_upload_sequence_s	int dsa_upload_sequence_s
ETB	int etb_etcom_start_download_ex	int etb_etcom_start_download_ex3
ETB	int etb_etcom_start_download_ex2	int etb_etcom_start_download_ex3
ETB	int etb_etcom_download_firmware	int etb_etcom_download_firmware_ex
DMD	int dmd_get_register_min_value	int dmd_get_register_min_value_int32
DMD	int dmd_get_register_max_value	int dmd_get_register_max_value_int32
DMD	int dmd_get_register_default_value	int dmd_get_register_default_value_int32
DMD	int dmd_get_parameter_min_value	int dmd_get_parameter_min_value_int32
DMD	int dmd_get_parameter_max_value	int dmd_get_parameter_max_value_int32
DMD	int dmd_get_parameter_default_value	int dmd_get_parameter_default_value_int32
DMD	ebool dmd_is_command_deprecated	N/A
DMD	ebool dmd_is_command_waiting	N/A
DMD	ebool dmd_is_parameter_jump_target	N/A
DMD	ebool dmd_is_parameter_l_value	N/A
DMD	int dmd_get_enum_value	int dmd_get_enum_value_int32
DMD	int dmd_get_enum_range	N/A
DMD	ebool dmd_is_enum_deprecated	N/A
TRA	int tra_receive_direct_cmd_e	nt tra_receive_direct_cmd_e_ex

15.5.2 .NET interface

Class	Deprecated method	Recommended alternative method
DsaDsmax	Class DsaDsmax	Class DsaMaster
DsaDsmaxGroup	Class DsaDsmaxGroup	Class DsaMasterGroup
DsaDrive, DsaDriveGroup	changeSetPoint	newSetPoint
DsaDrive, DsaDriveGroup, DsaMaster, DsaMasterGroup	setRegister	setRegisterInt32
DsaDrive, DsaMaster	getRegister	getRegisterInt32
DsaDrive, DsaDriveGroup, DsaMaster, DsaMasterGroup	setArray	setArrayInt32
DsaDrive, DsaMaster	getArray	getArrayInt32
DsaDrive, DsaDriveGroup, DsaMaster, DsaMasterGroup	getNewGate	N/A
DsaDrive, DsaDriveGroup, DsaMaster, DsaMasterGroup	setGate	N/A
DsaDrive, DsaDriveGroup, DsaMaster, DsaMasterGroup	clearGate	N/A
DsaDrive, DsaMaster	convertToIso	convertInt32ToIso
DsaDrive, DsaMaster	convertFromIso	convertInt32FromIso
DsaAcquisition	configTrigger	configImmediateTrigger (+ *other specific trigger methods*)
DsaAcquisition	configFrequency	configWithNbPointsAndTotalTime
DsaDrive, DsaMaster	quickRegisterRequest	quickRegisterInt32Request
DsaDrive, DsaDriveGroup, DsaMaster, DsaMasterGroup	resetError	resetErrorEx
DsaDrive, DsaDriveGroup, DsaMaster, DsaMasterGroup	resetErrorWithCheck	resetErrorWithCheckEx
DsaDrive, DsaDriveGroup, DsaMaster, DsaMasterGroup	executeSequence	executeSequenceInThread
DsaDrive, DsaDriveGroup, DsaMaster, DsaMasterGroup	stopSequence	stopSequenceInThread
DsaDrive, DsaDriveGroup	enableTrigger	enableTriggerEx
DsaDrive, DsaDriveGroup	trigger2dEnable	trigger2dEnableEx
DsaDrive, DsaDriveGroup	setPLSpeedFeedbackGain	setPLSpeedFeedbackGainEx
DsaDrive	getPLSpeedFeedbackGain	getPLSpeedFeedbackGainEx
DsaDrive, DsaDriveGroup	setPLProportionalGain	setPLProportionalGainEx
DsaDrive	getPLProportionalGain	getPLProportionalGainEx
DsaDrive, DsaDriveGroup	setPLIntegratorGain	setPLIntegratorGainEx
DsaDrive	getPLIntegratorGain	getPLIntegratorGainEx
DsaDrive, DsaDriveGroup	setCtrlSourceType	setCtrlSourceTypeEx
DsaDrive	getCtrlSourceType	getCtrlSourceTypeEx
DsaDrive, DsaDriveGroup	setCtrlSourceIndex	setCtrlSourceIndexEx
DsaDrive	getCtrlSourceIndex	getCtrlSourceIndexEx
DsaDrive, DsaDriveGroup	setCtrlOffset	setCtrlOffsetEx
DsaDrive	getCtrlOffset	igetCtrlOffsetEx
DsaDrive, DsaDriveGroup	setCtrlGain	setCtrlGainEx
DsaDrive	getCtrlGain	getCtrlGainEx
DsaDrive	getPositionCtrlError	igetPositionCtrlErrorEx
DsaDrive	getPositionDemandValue	getPositionDemandValueEx
DsaDrive	getPositionActualValue	getPositionActualValueEx
DsaDrive, DsaMaster	startUploadSequence	uploadsequence
EtbBus	etcomStartDownload	etcomStartDownloadEx
EtbBus	etcomDownloadFirmware	etcomDownloadFirmwareEx

Class	Deprecated method	Recommended alternative method
DmdData	getRegisterMinValue	getRegisterMinValueInt32
DmdData	getRegisterMaxValue	getRegisterMaxValueInt32
DmdData	getRegisterDefaultValue	getRegisterdefaultValueInt32
DmdData	getParameterMinValue	getParameterMinValueInt32
DmdData	getParameterMaxValue	getParameterMaxValueInt32
DmdData	getParameterDefaultValue	getParameterDefaultValueInt32
DmdData	isCommandDeprecated	N/A
DmdData	isParameterJumpTarget	N/A
DmdData	isParameterLValue	N/A
DmdData	getEnumValue	getEnumValueInt32
DmdData	getEnumRange	N/A
DmdData	isEnumDeprecated	N/A
DmdData	isRegisterDeprecated	N/A

16 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:

HEADQUARTER / SWITZERLAND	BELGIUM	CHINA
ETEL S.A. **Zone industrielle** **CH-2112 Môtiers** **Phone: +41 (0)32 862 01 00** **E-mail: etel@etel.ch** **http://www.etel.ch**	**HEIDENHAIN nv/sa** Pamelse Klei 47 1760 Roosdaal Phone: +32 54 34 31 58 E-mail: sales@heidenhain.be	**DR. JOHANNES HEIDENHAIN (CHINA) Co., Ltd** No. 6, Tian Wei San Jie, Area A, Beijing Tianzhu Airport, Industrial Zone Shunyi District, Beijing 101312 Phone: +86 400 619 6060 E-mail: sales@heidenhain.com.cn
CZECH Republic	**FRANCE**	**GERMANY**
HEIDENHAIN s.r.o. Dolnomecholupská 12b 102 00 Praha 10 - Hostivar Phone: +420 272 658 131 E-mail:heidenhain@heidenhain.cz	**HEIDENHAIN FRANCE SARL** 2 avenue de la cristallerie 92310 Sèvres Phone: +33 (0)1 41 14 30 09 E-mail: sales@heidenhain.fr	**DR. JOHANNES HEIDENHAIN GmbH** Technisches Büro Südwest II Verkauf ETEL S.A. Schillgasse 14 78661 Dietingen Phone: +49 (0)741 17453-0 E-mail: tbsw.etel@heidenhain.de
GREAT-BRITAIN	**ISRAEL (Representative)**	**ITALY**
HEIDENHAIN (GB) Ltd. 200 London Road, Burgess Hill, West Sussex RH 15 9RD Phone: +44 (0)1444 247711 E-mail: sales@heidenhain.co.uk	**MEDITAL COMOTECH Ltd.** 36 Shacham St., P.O.B 7772, Petach Tikva Israel 4951729 Phone: +972 3 923 3323 E-mail: comotech@medital.co.il	**ETEL S.A.** Piazza della Repubblica 11 28050 Pombia Phone: +39 0321 958 965 E-mail: etel@etelsa.it
JAPAN	**KOREA**	**SINGAPORE**
HEIDENHAIN K.K. Hulic Kojimachi Bldg. 9F 3-2 Kojimachi, Chiyoda-ku Tokyo - 102-0083 Phone: +81 3 3234 7781 E-mail: sales@heidenhain.co.jp	**HEIDENHAIN KOREA Ltd.** 75, Jeonpa-ro 24beon-gil, Manan-gu, Anyang-si Gyeonggi-do, 14087, Korea Phone: + 82 31-380-5304 E-mail: etelsales@heidenhain.co.kr	**HEIDENHAIN PACIFIC PTE. LTD** 51 Ubi Crescent, Singapore 408593 Phone: +65 6749 3238 E-mail: info@heidenhain.com.sg
SPAIN (Representative)	**SWEDEN**	**SWITZERLAND**
Farresa Electronica, S.A. C/ Les Corts, 36 bajos ES-08028 Barcelona Phone: +34 93 409 24 91 E-mail: farresa@farresa.es	**HEIDENHAIN Scandinavia AB** Storsätragränd 5 127 39 Skärholmen Phone: +468 531 93 350 E-mail: sales@heidenhain.se	**HEIDENHAIN (SCHWEIZ) AG** Vieristrasse 14 CH-8603 Schwerzenbach Phone: +41 (0)44 806 27 27 E-mail: verkauf@heidenhain.ch
TAIWAN	**THE NETHERLANDS**	**UNITED STATES**
HEIDENHAIN CO., LTD. No. 29, 33rd road, Taichung Industrial Park Taichung 40768, Taiwan, R.O.C. Phone: +886 4 2358 8977 E-mail: info@heidenhain.tw	**HEIDENHAIN NEDERLAND B.V.** Copernicuslaan 34 6716 BM Ede Phone: +31 (0)318 581800 E-mail: verkoop@heidenhain.nl	**HEIDENHAIN CORPORATION** 333 E. State Parkway Schaumburg, IL 60173 Phone: +1 847 490 1191 E-mail: info@heidenhain.com

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.