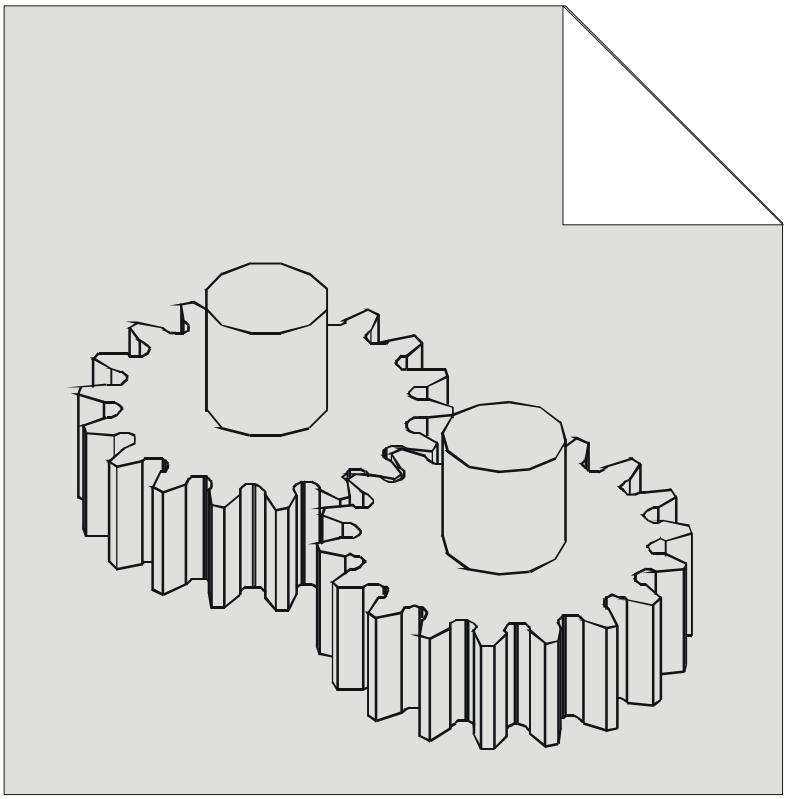
MOTION TECHNOLOGY

ETEL Device Interface



EDI User's Manual

EDI 902 VER A

HEADQUARTERS

ETEL S.A.
CH-2112 Môtiers - Switzerland
Tel : +41 32 862 01 23
Fax : +41 32 862 01 01
E-mail : etel@etel.ch
http://www.etel.ch

ETEL GmbH

Schillgasse 14
D-78661 Dietingen - Germany
Tel : +49 741 1745 30
Fax : +49 741 1745 399
E-mail : etel@etelgmbh.de

ETEL, Inc.

333 E. State Parkway,
USA - Schaumburg, Illinois 60173-5337
Tel : +1 847 519 3380
Fax : +1 847 490 0151
E-mail : info@etelusa.com
Internet : http://www.etelusa.com

THIS PAGE IS INTENTIONALLY LEFT BLANK

Table of contents

Record of revisions, document # EDI x

Document revisions		
Issue (x)	**Date**	**Modified**
Ver A	19/11/2001	First edition

ETEL documentation concerning the EDI

- **User's Manual** **EDI principle & operation** **# EDI 902 A**

1. Introduction

1.1 Glossary

Technical acronyms

Abbreviation	Definition
API	Application Program Interface
DLL	Dynamic Link Library
EDI	ETEL Device Interface
EB	ETEL-Bus
EBL	ETEL-Bus-Lite
ISO	International Standards Organization
TEB	Turbo-ETEL-Bus
URL	Uniform Resource Locator

The ETEL Device Interface (EDI) is a set of libraries which enable the communication with the ETEL products (DSA2-P, DSB2-P, DSC2-P and DSMAX) and the access to their functionalities. All the functions contained in those libraries are often called API (Application Program Interface).

The main role of that set is to:

- Generate the communication through several existing communication buses such as the serial port, the ISA bus, and the LAN.
- Give the user a group of functions allowing him to have access to all the functionalities of the ETEL products.
- Give an homogeneous interface whatever the product and the firmware version.
- Manage the conversion of the products' internal units (increments) in order to enable the user to work with conventional units (ISO).

Those libraries are totally developed in C and can be used from all the other common programming languages such as C++, Visual Basic and LabVIEW. They can be used from other languages but most of the time the interface needs to be developed.

Windows 9x/NT/2000	DLL	File .dll	MS Visual C/C++, Borland C++ Builder, LabVIEW, Visual Basic, Java
QNX4	Static library	File .lib	Watcom C

Most of the applications call for the high level library called DSA which afterwards call for the other libraries of the EDI package. Despite its name, this library is able to manage the DSA2-P as well as the DSB2-P, the DSC2-P and the DSMAX.
Currently, the DSA is the library which offers the most important number of interfaces allowing its use from a large number of programming languages. It's also the only library currently supported by ETEL.

1.2 Contents

This document presents all the libraries of the EDI package by putting the emphasis on the DSA library. This library is the only useful one during the programming of the drives (DSA2-P, DSB2-P, DSC2-P, DSCD-P) from a PC.

The other libraries are mainly used by the DSA library and can be directly used only for special operations.

1.3 Dictionary

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

DLL	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 is starting up or while it is running. The process then executes the functions in the DLL which means "dynamic link library".
Drive	An ETEL digital servo amplifier such as DSA2-P, DSB2-P, DSC2-P or DSCD-P.
Device	A device can be a drive (DSA2-P, DSB2-P or DSC2-P) or a DSMAX.
OS	Operating System. It can be Windows 9x/NT/2000, QNX4 or others.
Object	When we talk about 'object', we refer to the definition widely used in the object-oriented programming. When we work in C, an object is nothing more than a structure containing a series of information. The classical example of C object is the FILE structure of the standard library.
Increment	ETEL's devices use a large number of physical quantities (position, speed, acceleration, force, time, electric current, ...). These quantities are often represented by 32 bits integers with units peculiar to the devices (UPI, USI,...). These units are called 'increments', then the 'representation by increments' term is used to represent the physical quantities.
ISO units	The international system of units is used to give 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.

1.4 DLL (Dynamic Link Library)

The EDI package for Windows 9x/NT/2000 is a set of 'dynamic link libraries' commonly known as DLL.

This kind of library is a standard for Windows and all the programmers have 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.
The main points characterizing a DLL are:

- a Windows standard
- its format is known by all the compilers
- A DLL is dynamically linked. It allows the user to modify or to update it without compiling again the application using it.

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 goes and gets them.

1.5 Supported Operating Systems

At the moment, the EDI package is available for Windows 9x/NT/2000 and QNX4

Concerning Microsoft's operating systems, it is strongly advisable to use Windows NT 4.0 or 2000. These products are actually reliable and robust. On the other hand, Windows 95 and 98 have showed weaknesses in the same fields, especially during the use of the serial port as a communication bus with ETEL's products.

1.6 Supported Communication Buses

At the moment, the EDI package stands the following communication bus:

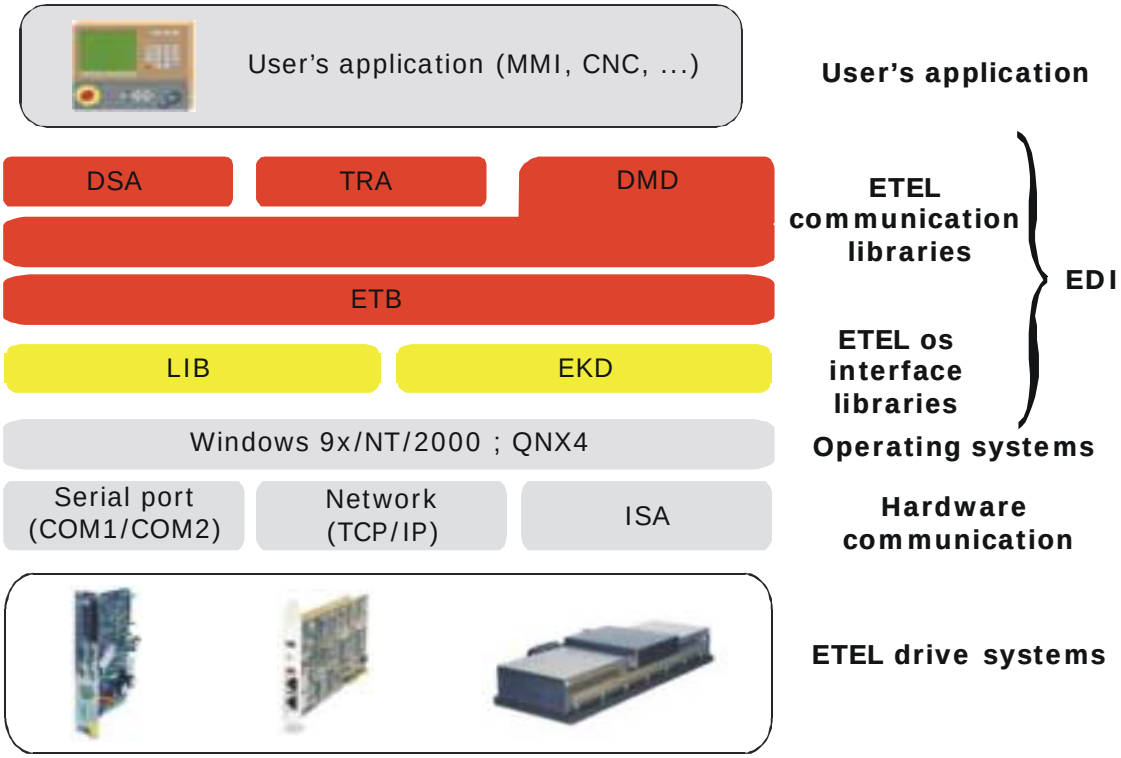
- ISA bus (during the use of the DSMAX)
- Serial port (EBL / EBL2)
- TCP/IP link (during the use of the ETND)

WARNING: the use of the serial port as a communication bus presents very important limitations. For applications requiring a good reliability and performances, this type of communication is not recommended.

Its use is mainly planned to start and set up the drives (via ETEL Tools). Nevertheless it can be used for fairly non-demanding applications. Anyway, please be careful of the following points:

- The serial port is based on the RS232 (EIA232) norm. **This norm is not suitable for industrial environments** because too sensitive to the disturbances and ground loops**. It is better to work with a RS422 or RS485 connection.** On the market, it is possible to find PC board with RS485 interface or RS232-RS485 adapter which can be plugged on a serial port. Most of ETEL's products have a RS422/485 interface. It is then not necessary to have an adapter on the drive.
- Most of the PC's serial ports have a 16 bytes input buffer (FIFO). As the EBL and EBL2 connections use a software flow control, the 16 bytes buffer fills up quickly and an overflow can be easily reached. The operating system has an important function here. Some tests highlighted that Windows 95 and 98 do not answer fast enough to the requests of interrupt and then lose data when the CPU is really busy. **It is then better to use a RS232 or RS485 board whose buffer is bigger than 16 bytes.** The use of Windows 95 or 98 is also not recommended.
- The serial port does not have very high performances. Despite the use of a 115200 bauds rate, the transactions between the PC and the drives are slow.
- During the use of the serial port, the status are not managed in real time and then theirs efficiency is incredibly reduced.
- The experience has shown that the power management on laptops often induces a malfunctioning of the serial port.

2. Software structure

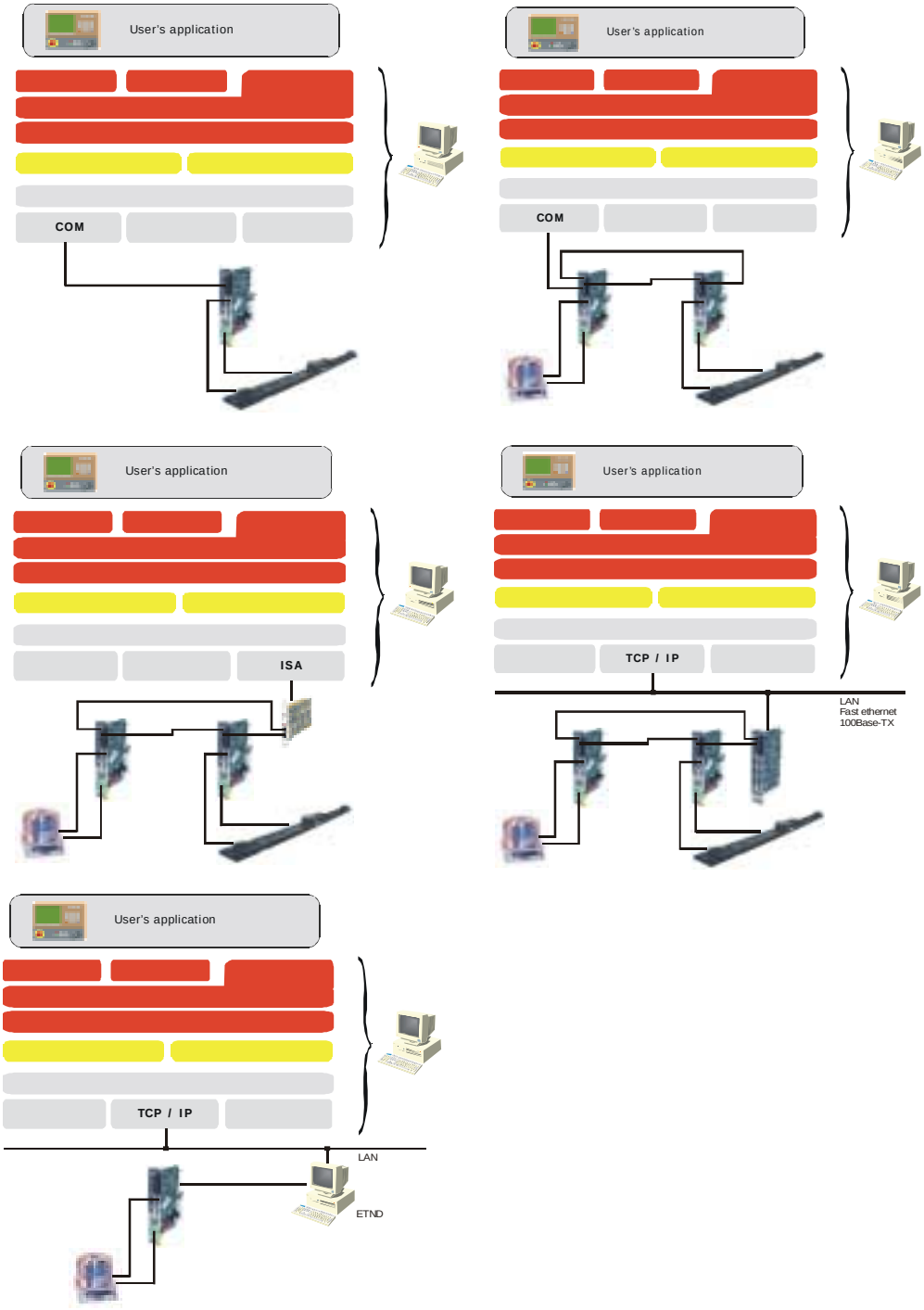


2.1 Main characteristics

The main characteristics of the software are:

- Implemented in ANSI C
- Can be used from C, C++ (Visual C++ and Borland C++ Builder), LabVIEW (with some restrictions), Visual Basic (with some restrictions)
- Common source files for multiple platforms and operating system
- C++ wrapper classes included in header files
- Thread-safe implementation
- Object-oriented architecture

2.2 Communication configuration



2.3 Library's rules

2.3.1 DSA Library

- Manages single drive, drives group, DSMAX and interpolation group objects.
- Generic functions for using all the functionalities of the drive(s) and the DSMAX.
- Performs conversions between the device internal units (increments) and the ISO units.
- Transparent to communication configuration.

- Manages devices on several communication buses at the same time.
- Specific functions to set and get principal drive registers.
- Specific functions to execute most commands used by the drive(s) and the DSMAX.
- Asynchronous and synchronous functions.

2.3.2 TRA Library

- Manages the translation between the ETEL language (sequence and terminal commands) and the EBL or EBL2 records internally used by the drives.

2.3.3 ETB Library

- Manages the communication buses and logical ports.
- Handles the EBL, EBL2, TEB and TCP/IP transparently.
- Handles the normal and boot communication.
- Handles the status update (see §3.11).
- Multiple logical port message queuing.
- Registering of event handlers (callback).

2.3.4 DMD Library

- 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 drive database.
- Stores information for each device and firmware version.

2.3.5 LIB Library

- Thread management and realtime functions.
- Serial port functions.
- Timer functions.
- Queue manipulation functions.

2.3.6 EKD Library

- Manages all accesses to the hardware.
- Manages the physical address mapping.
- Manages interrupt service routines.
- Allows the access to the PnP informations.

2.4 Versions numbering

All the ETEL softwares are endowed with a version number identified as follow:

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

For example, a version number can be: 1.00A, 3.15B, 2.00D, etc... An "X" can follow the version number if this version is under construction. A special numbering is used for the preliminary versions (alpha or beta version).

Each EDI package and library have their own version number. There is no direct link between both version numbers.

For the libraries, the first two digits of the version number are present in the file name of this library.

To know the exact version of a library, open windows explorer and right click on the wanted file to select 'Properties'.

Each library also includes a function which gives the version number. For the DSA library, the function is called: dsa_get_version(), and for the ETB library, it is called: etb_get_version(), etc ...

2.5 Files for Windows

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

dsa20.h	Header file necessary during the compilation
dsa20c.lib	Static library necessary during the link
dsa20c.dll	Dynamic link library necessary during the execution

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

dsa20cd.lib	Static library, debug version
dsa20cd.dll	Dynamic link library, debug version

The files contain the name of the library followed by a number which takes back the first two digits of the version. 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 C++. If you use Borland C++ Builder, these libraries can be converted to the OMF format thanks to the coff2omf.exe command. Otherwise, the following libraries are available:

dsa20bc.lib	Static library for Borland C++ Builder
dsa20bcd.lib	Static library for Borland C++ Builder, debug version

To use the DSA library from Visual Basic, the following files are available:

dsa20.bas	Visual Basic file to be included in the project
dsa20vb.dll	Interface library between Visual Basic and DSA
dsa20vbd.dll	Interface library between Visual Basic and DSA, debug version

To use the DSA library from LabVIEW, a VIs collection is available.

For the libraries other than the DSA one, the files are more or less the same. These libraries are internally used by the DSA library or for special applications. Not all the interfaces and the files are available for each library. For example, the TRA library does not have any Visual Basic interface and then the tra10vb.dll file does not exist. On the other hand, the following files are available:

tra10.h	Header file

tra10c.lib	Static library
tra10cd.lib	Static library, debug version
tra10c.dll	Dynamic link library
tra10cd.dll	Dynamic link library, debug version

Normally, only the DSA library is directly used by the customers. ETEL does not give any support concerning the use of other libraries such as TRA and ETB.

2.6 Files for QNX4

In the QNX version of the EDI package, there is no DLL file. All the functions are stored in static libraries. Then, the code of the functions is contained in the .lib files. With QNX4, the libraries can be used from a C program compiled with the Watcom compiler.

3. Use of the DSA20 in C

This chapter describes the functions allowing the user to establish the communication with an ETEL device and to realize different types of operation.

All the examples described in this chapter are written in C. It is advisable to have a good knowledge of programming and C language to understand their functioning.

Nevertheless, this chapter is really useful for the ones who would like to program with other languages such as Visual Basic because the names of the functions and their parameters do not change a lot in the different programming languages. The structure of the library and the way to work are also the same.

3.1 Errors 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 dsa20.h. For the DSA library the possible error codes are as follow:

#define	Error code	Comment
DSA_EBADDRVVER	-325	A drive with a bad version has been detected
DSA_EBADIPOLGRP	-327	The ipol group is not correctly defined
DSA_EBADPARAM	-322	One of the parameter is not valid
DSA_EBADSTATE	-324	This operation is not allowed in the state
DSA_EBUSERROR	-313	The underlayer of ETEL-Bus is not working properly
DSA_EBUSRESET	-314	The underlayer of ETEL-Bus is performing a reset operation
DSA_ECANCEL	-319	The transaction has been cancelled
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_EINTERNAL	-316	Some internal error in the ETEL software
DSA_ENOACK	-312	No acknowledge from the drive
DSA_ENODRIVE	-320	The specified drive does not respond
DSA_ENOTIMPLEMENTED	-326	The specified operation is not implemented
DSA_EOPENPORT	-321	The specified port cannot be opened
DSA_ESYSTEM	-315	Some system resources return an error
DSA_ETIMEOUT	-310	A timeout has occurred
DSA_ETRANS	-318	A transaction error has occurred

The user can test the smooth functioning of a DSA function in the following way:

```
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 textual error message in the form of a character string. Here is the prototype:

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

Here is a use example of 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");
...
```

3.2 Creation of objects

The DSA library as all the other libraries of the EDI package, is totally implemented in C by respecting the object-oriented philosophy. The work is done with objects represented by simple structures in C.

Then, an object corresponds to each device (DSA2-P, DSB2-P, DSC2-P, DSCD-P or DSMAX), and all the functions concerning this device receive a pointer on this object (this structure) as parameter. The content of the structure is not visible for the user.

To illustrate that, look at the functions of the standard C library which allow the access to the 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 on this object. It is the case for fprintf() which receives as first parameter the pointer on the FILE structure.

To have access to a drive (DSA2-P, DSB2-P, DSC2-P or DSCD-P), a pointer on the DSA_DRIVE object must be created in the following way:

```
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 at the NULL value.

Once the pointer declared, the object must be created and the pointer affected. It is done in the following way:

```
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 in the following way:

```
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 DSMAX, is done in a similar way:

```
DSA_DSMAX *dsmax1 = NULL;
int err ;

err = dsa_create_dsmax(&dsmax1);
if (err != 0) {
    printf("cannot create drive object\n");
    exit(1);
}
```

3.3 Opening of the communication

Once the drive object created, a physical drive must be associated with it. The communication bus used to communicate with the drive as well as the physical address of the drive (axis number) must be indicated. For example, if the user wants that the drive1 object corresponds to the axis 12 which is connected to the serial port of the PC, the following function must be called:

```
err = dsa_open_u(drive1, "etb:com1:12");
```

This function will first open the communication bus on the condition that it is not already opened and then will memorize the axis number in the drive1 object. The second parameter is a character string (a URL= Uniform Resource Locator) which identifies the bus and the drive used. Here is the syntax:

<protocol> : <communication bus> : <axis number>

Here is the list of the value for the three above-mentioned fields:

Field	Value	Description
<protocol>	etb	ETEL-Bus protocol
<communication bus>	com1 com2 com1:9600 ... isa:0xD0000,10	Communication through the serial port 1 Communication through the serial port 2 Communication through the serial port 1 at 9600 bauds Communication through the DSMAX (ISA multi-axis board) (see the DSMAX user's manual for details)
<axis number>	0 1 ... 30 *	Axis number 0 or node number 0 Axis number 1 or node number 1 Axis number 30 or node number 30 Multi axis board (DSMAX)

At the moment, the only protocol used is ETEL-Bus. The list of the communication bus is not exhaustive. To use special buses like the TCP/IP, please refer to the corresponding application note.

To establish the communication of a DSA_DSMAX object, the same dsa_open_u() function is used. In the description chain (URL), a star will be indicated instead of the axis number.

3.4 Specific functions

Once the object created and the communication opened, operations can be executed on the devices. For example, to switch on the drive, 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_DSMAX object which represents the device on which the operation will be done. The last parameter is always the 'timeout' that is to say the maximum time which is authorized to effectuate the operation. This time is always given in millisecond.

The call of this function ends when the operation is finished or when the timeout has elapsed. In the above example, the function ends when the motor is switched on and the control part (regulation) is activated, or when the process lasts more than 10 seconds. In that case, the function returns an error.

For the operations where the execution time does not depend on the drive configuration or on the application, the default timeout can be used by giving the DSA_DEF_TIMEOUT (which will be explained later on).
Some other functions need more parameters. For example, it is the case for the function which starts a movement and also needs 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 that type of function, the parameters representing a physical quantity are always given in ISO units. For example, in the previous example, 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:

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	

Here are some code lines which show the use of specific functions:

```
/* clear all errors on the drive */
err = dsa_reset_error_s(drive1, DSA_DEF_TIMEOUT);
if (err != 0)
    exit(err);

/* switch the power on */
err = dsa_power_on_s(drive1, 10000);
if (err != 0)
    exit(err);

/* start the homing procedure */
err = dsa_homing_start_s(drive1, 10000);
if (err != 0)
    exit(err);

/* wait for the end of the homing procedure */
dsa_wait_movement_s(drive1, 60000);

/* start a movement */
dsa_set_traget_position(drive1, 0, 0.5, DSA_DEF_TIMEOUT);
```

```
/* wait for the end of the movement */
dsa_wait_movement_s(drive1, 60000);
```

A list of the available functions is given in the appendix (see §4.). If the user is used to program sequences in the ETEL language or if he knows the commands of the ETEL terminal, he will find the correspondence between these commands and the functions of the DSA library.

3.5 Generic functions

All the operations that can be done on a device come down to three operations:

- To send a command with zero, one or several parameters
- To set the value of a register
- To read the value of a register

On top of these three basic operations, there is the status management which will be detailed in §3.11.
Most of the operation 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 generic functions doing these three operations. It is really useful to have access to the functionalities for which there is no specific function. For example, the DSA library does not have any function to reset the drive (RSD). To do so, the user has to use the generic functions.

3.5.1 Units conversion

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, the ETEL devices work with internal units often called «increments». Here is a table illustrating the differences between these two types of units:

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

When using the specific functions, the DSA library does automatically the necessary units 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 increments or the 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 is an example:

- The user wants to modify the K210:0 register of a drive
- K210:0 represents the position of a linear motor
- This quantity in increments (in the drive) is given in UPI
- This quantity in ISO units is given in meters [m]

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

```
err = dsa_set_register_s(
    drive1,          /* grp: destination device */
    2,               /* typ: 2 = K register */
```

```
    210,              /* idx: register index */
    0,                /* sidx: register subindex */
    240000,           /* value: register value in UPI */
    DSA_DEF_TIMEOUT   /* 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 */
    210,              /* idx: register index */
    0,                /* sidx: register subindex */
    0.12,             /* value: register value in [m] */
    DMD_CONF_UPI,     /* conv: drive unit (UPI) */
    DSA_DEF_TIMEOUT   /* timeout: default timeout */
);
```

In the generic functions, the parameters representing a quantity in increments are called "Val" and are "long" type parameters. The parameters representing a quantity in ISO units are also called "Val" but are "double" type parameter and are preceded by a "conv" parameter of "int" type. The following values can be ascribed to this parameter:

Values (#define)	Device's unit	Description
DMD_CONV_UPI	UPI	user position increment
DMD_CONV_USI	USI	user speed increment
DMD_CONV_UAI	UAI	user acceleration increment
DMD_CONV_DPI	DPI	drive position increment
DMD_CONV_DSI	DSI	drive speed increment
DMD_CONV_DAI	DAI	drive acceleration increment
DMD_CONV_C13	C13	current 13bit range
DMD_CONV_C14	C14	current 14bit range
DMD_CONV_C15	C15	current 15bit range
DMD_CONV_C29	C29	current 29bit range
DMD_CONV_CUR2	CUR2	i2, dissipation value
DMD_CONV_CUR2T	CUR2T	i2t, integration value
DMD_CONV_STI	STI	slow time interrupt
DMD_CONV_FTI	FTI	fast time interrupt
DMD_CONV_CTI	CTI	current time interrupt
DMD_CONV_EXP10	EXP10	ten power factor
DMD_CONV_UFTI	UFTI	user friendly time increment
DMD_CONV_AVI	AVI	analog voltage increment
DMD_CONV_VOLT		
DMD_CONV_UFPI	UFPI	user friendly position increment
DMD_CONV_UFSI	UFSI	user friendly speed increment
DMD_CONV_UFAI	UFAI	user friendly acceleration increment

To use the #define's values, the 'dmd10.h' file must be included at the beginning of the source file in the following way:

```
#include <dmd10.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 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.

3.5.2 Sending of commands

Each device is able to execute a large number of command which does all sorts of operations.

Each command has a number and 0, 1 or several parameters. These parameters can belong to one of the following categories:

- Parameters with unit: they represent physical quantities such as positions, speeds, times, etc... They can be given in increments or ISO units. In increments, they are coded in 32 bits integers. In ISO units, they are coded in double precision floating points.

- Parameters without units: they often represent non-physical quantities such as the number of the axis, the error number, the digital input/output state, etc... These quantities are coded in 32 bits integers.

- Special parameters: they represent several information, coded in a 32 bits integer. This kind of parameters is not developed in this manual.

Here is an example of command that a drive 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.

Given that the parameters in increments, the parameters without unit, and the special ones are coded in 32 bits integers, they are not differentiated at the level of the DSA library functions.

At the DSA library level, two types of parameters are differentiated:

- Long type parameters (32 bits integer)
- Double type parameters (floating point double precision)

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 long type parameter
dsa_execute_command_i_s()	One double type parameter
dsa_execute_command_dd_s()	Two long type parameters
dsa_execute_command_id_s()	Two parameters: the first of double type and the second of long type
dsa_execute_command_di_s()	Two parameters: the first of long type and the second of double type
dsa_execute_command_ii_s()	Two parameters of double type
dsa_execute_command_x_s()	Any number and type of parameters

If the user wants to send the command number 119 (HLO) to a drive, he must use the dsa_execute_command_s() function as follows:

```
err = dsa_execute_command_s(
   drive1,            /* 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, if the user wants to send the command number 109 (AXI) with the parameters 1203 and 3, he must proceed as follows:

```
err = dsa_execute_command_dd_s(
   drive1,            /* 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 */
);
```

The complete prototypes of these generic functions are in §4.2.

In the meaning of the different parameters, the first one is always the DSA_DRIVE or DSA_DSMAX object which represents the device on which the operation has to be done. 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_DSMAX 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 §3.5.1
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 the number of parameters is variable. First, the user must create a DSA_COMMAND_PARAM type array whose the number of elements corresponds to the number of the command's parameters. Each element in that table must be then assigned with the type, the value, and the conversion type of the corresponding parameter. Once the array created and assigned, the dsa_execute_command_x_s() command must be called by giving it as a parameter, the pointer and the size of this array.

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

```
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(dsmax, 1025, params, 5,
  FALSE, FALSE, DSA_DEF_TIMEOUT);
```

3.5.3 Allocation and reading of registers

Each device has several registers families. The three families usually called are:

Family	Name	Family number (typ)
Parameters (PPK)	K0, K1, K2, ...	2
Monitorings register (read only)	M0, M1, M2, ...	3
User variables	X0, X1, X2, ...	1

The DSA library has a specific function to read and assign each register widely used. On top of that, it offers 4 generic functions which are able to assign and read any register:

Function	Description
dsa_get_register_s()	Allows the reading of an increments value of any register
dsa_get_iso_register_s()	Allows the reading of an ISO units value of any register
dsa_set_register_s()	Allows the allocation of an increments value of any register
dsa_set_iso_register_s()	Allows the allocation of an ISO units 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_s(
  drive1,            /* 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_s(
  drive1,            /* 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 K211:0 parameter, multiplies it by 10 and stores it again in the K211:0 parameter. The K211:0 parameter is in USI unit.

```
double speed;        /* in m/s */

err = dsa_get_iso_register_s(
  drive1,            /* grp: destination device */
  2,                 /* typ: 2 = K 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(
  drive1,            /* grp: destination device */
  2,                 /* typ: 2 = K 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 */
);
```

In §4.2.3 is the whole prototypes of these generic functions. Here in details is the meaning of their parameters:

Parameter	Name	Description
First parameter	dev / grp	DSA_DRIVE or DSA_DSMAX object
Specific parameters	typ	1 = user variable (X) 2 = parameter (K) 3 = monitoring register (M)
	idx	The register index
	sidx	The register subindex
	value	Value to assign to the pointer on the variable which receives the value in return. It can be a long type (long*) or a double type (double*) parameter 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.
	kind	It is used by the functions which read the registers to indicate if the user wants to read the current, the maximum, the minimum or the default value or the unit type used in the device. The following values are then possible: DSA_GET_CURRENT, DSA_GET_MIN_VALUE, DSA_GET_MAX_VALUE, DSA_GET_DEF_VALUE, DSA_GET_CONV_FACTOR
Last parameter	timeout	Maximum time of execution

3.6 Groups

In the previous paragraphs, 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 on an object which includes all the information of the concerned drive or DSMAX.

The user often needs to send a command or to do an operation on several drives at the same time. It is necessary when the user wants to start synchronized movements. For that purpose, the DSA library has the possibility to create groups of devices. Afterwards, the user can do operations on these groups instead of doing it on a simple device.

The DSA library enables the definition of the following groups:

Group	Type	Description
Device group	DSA_DEVICE_GROUP	Group of several drives or DSMAX
Drive group	DSA_DRIVE_GROUP	Group of several drives
DSMAX group	DSA_DSMAX_GROUP	Group of several DSMAX
Interpolation group	DSA_IPOL_GROUP	Group of several drives on which an interpolated movement can be done

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

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

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

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

Compared with 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 put in 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 in the following way:

```
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 already created. Instead of allocating a drive, 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 drive has been created. Then, only the DSA_DRIVE, DSA_DRIVE_GROUP, or the DSA_IPOL_GROUP objects' type can be assigned to it. If the user wants to include the DSMAX in a group, the DSA_DEVICE_GROUP group type must be created.
A group of drives can include drives which are not on the same communication bus. For example, it is possible to have in the same group, a drive connected to COM1 and another one connected to COM2. On the other hand, in a DSA_IPOL_GROUP, the drives are only accessible via the same DSMAX which means that there are connected to the same Turbo-ETEL-Bus (TEB) (See the DSMAX 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:

```
err = dsa_power_on_s(group1, 10000);
```

On the other hand, it is not possible to read a register on a group and then the following example cannot be done:

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

See §3.10 for more details about the hierarchy of the different objects and which type of object is accepted by each DSA library function.

3.7 Closing and destruction of the objects

Each communication with a device is preceded by two basic operations: the creation of an object and the opening of the communication.

These two functions assign resources to the operating system. During the creation of an object or a group, memory is assigned. During the opening of the communication, the use of the serial port, the interruption line or other resources peculiar to the communication bus used are assigned. These resources must be given back to the system as soon as there are not used any more. It is normally done just before leaving the application.

The DSA library has the dsa_close() function to close the communication and the dsa_destroy() function to destroy an object or a group. The closing must be done before the destruction. Here is an example:

```
DSA_DRIVE *x = NULL;
DSA_DRIVE *y = NULL;
DSA_DRIVE_GROUP *xy = NULL;

dsa_create_drive(&x);
dsa_create_drive(&y);
dsa_create_drive_group(&xy, 2);
dsa_add_group_item(x);
dsa_add_group_item(y);

dsa_open_u(x,...);
dsa_open_u(y,...);
```

```
... operations on the drives ...

dsa_close(x);
dsa_close(y);
dsa_destroy(&xy);
dsa_destroy(&x);
dsa_destroy(&y);
```

Remark: The dsa_destroy function asks as parameter the address of the pointer on the object (&x, &y and &z) and not the pointers themselves (x, y and xy). In this way, once the object destroyed, its address is assigned to NULL.

3.8 Synchronous functions and timeouts

Until now, only synchronous functions have been used. These functions are called by the user and their execution ends once the desired operation is finished.

For example, when the dsa_get_register_s() function is called, it asks the device the value of a register and waits for the answer. Once the device has returned the value to it, the function ends by stepping down in favour of the user.

The synchronous functions stop then the execution of the program until the end of the current operation. It avoids the user to do another operation before the current one is finished. The limit can be passed by using several threads or asynchronous functions (see §3.9).

In the DSA library, the synchronous functions are characterized by a '_s' at the end of their name and by a 'timeout' parameter which is always the last one.

The 'timeout' parameter allows the limitation of the maximum time that the function uses to end an operation. After this time, if the operation is not finished yet, 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 do the operation, the INFINITE value can be assigned to the timeout parameter.

If the user wants to execute a simple operation with a time of execution which does not depend on the application, he can use the timeout by default. To do so, the DSA_DEF_TIMEOUT value must 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 timeout by default does not make sense.

3.9 Asynchronous functions and callbacks

During the development of an application, we mainly use all the previous synchronous functions. Nevertheless, in a few cases the use of synchronous functions is not ideal or requires more threads. For example it is the case when the user wants to monitor the status of a drive at the same time than he waits for the end of a movement. In this case, the use of asynchronous functions is interesting. As mentioned above, the synchronous functions have a name ending with '_s' and a last parameter called 'timeout'. An asynchronous function corresponds to every synchronous function. Their name ends with a '_a' instead of a '_s' and the last parameters are 'handler' and 'param' instead of 'timeout'.

The asynchronous functions start an operation without waiting for the end. Their execution is then really fast. After having started the operation, the user can execute other operations without starting another thread.

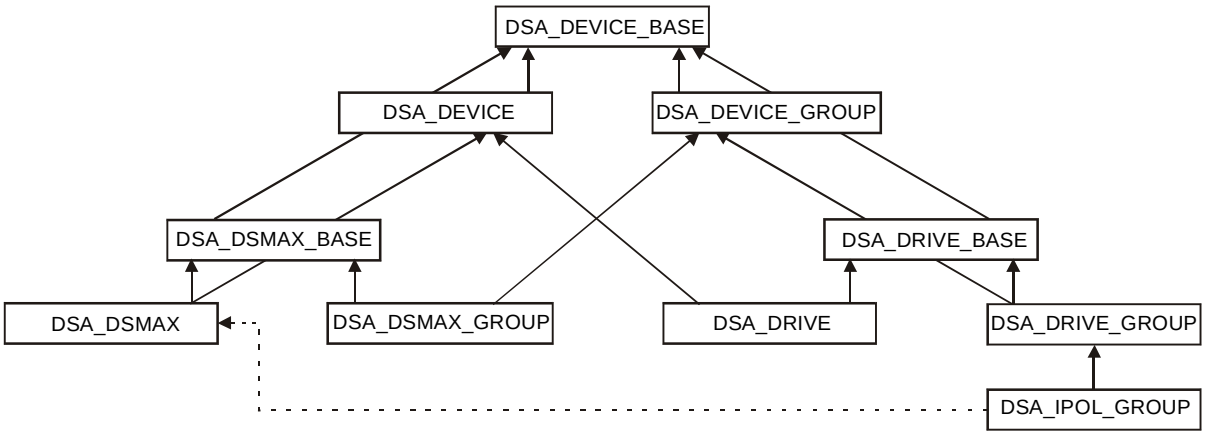
When the user calls an asynchronous function, he must give it a handler as parameter which is a pointer on a function often called 'callback'. This function is called by a thread within the library once the operation is finished.

For example, if the user calls the dsa_get_register_a() function, it ends before the value of the register is

returned by the device. Once the value returned, the library calls the handler by giving it the value as parameter.

Besides the handler, the user has the 'param' parameter that he can use as he wishes. This parameter is not interpreted by the library and is given to the handler when it is called.

3.10 Heritage



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

For example, the power of a drive can be switched on and off, but it is not possible to do it with a DSMAX. The dsa_power_on_s() will then accept the objects of DSA_DRIVE and DSA_DRIVE_GROUP type but not the DSA_DSMAX type.

In the same way, the reading of a register can be done on only one device at the same time. The dsa_get_register_s() function will then accept the objects of DSA_DRIVE and DSA_DSMAX type 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 below.

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_s().

From these relations of heritage and the prototype of the function, it is easy to know to which object is applied a given function.

3.11 Status

3.11.1 Principle

Each ETEL's device has a set of information which allows the user to know what the state of the device is. 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.

Here is the list of the states common to all the devices:

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	Defined by the user
User 1	
User 2	
User 3	
User 4	
User 5	
User 6	
User 7	
User 8	Defined by the user; not available with DSA2-P and DSB2-P
User 9	
User 10	
User 11	
User 12	
User 13	
User 14	
User 15	

According to the type of device and communication bus, the user can have extra status. For example, if a DSA2-P or DSB2-P is used the user will have lots of additional status. On the other hand, the user will only have 8 user status and limitations will occur because of the use of the EBL communication.

The main special feature of the status is their update in real time. The PC always knows the status of all devices. If the status of a device changes, the PC is immediately informed. Therefore, the user knows all the time the status of the devices. To do so, he just has to call the dsa_get_status() function which simply returns the status stored in the PC. Its execution is then really fast because no external communication with the PC is done.

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

Here is the list of these functions:

dsa_get_status()	Returns the status of a device
dsa_wait_status_equal_s()	Waits for a device to be in a given state
dsa_wait_status_not_equal_s()	Waits for a device to quit a given state
dsa_grp_wait_and_status_equal_s()	Waits for all the devices of a group to be in a given state
dsa_grp_wait_and_status_not_equal_s()	Waits for all the devices of a group to quit a given state
dsa_cancel_status_wait()	Cancels the current waiting of the status of a device or a group of devices

Asynchronous versions are added to these functions. There are:
dsa_wait_status_equal_a(), dsa_wait_status_not_equal_s(), dsa_grp_wait_and_status_equal_a() and dsa_grp_wait_and_status_not_equal_a().

The whole prototype of these functions is given in §4.2.4.

3.11.2 Functioning

The DSA library stores the status in a structure called 'DSA_STATUS'. Here is the definition:

```
typedef union DsaStatus {
  size_t size;                /* Size of this structure */
  struct {
     size_t size;
     dword sw1;
     dword sw2;
  } raw;
  struct {
     size_t size;
     unsigned :3;
     unsigned present:1;    /* the drive is present */
     unsigned moving:1;     /* the motor is moving */
     unsigned in_window:1;  /* position is in window */
     unsigned :2;
     unsigned sequence:1;   /* a sequence is running */
     unsigned :1;
     unsigned error:1;      /* fatal error */
     unsigned trace:1;      /* trace aquisition in progress */
     unsigned :4;
     unsigned :7;
     unsigned warning:1;    /* global warning */
     unsigned :8;
     unsigned :4;
     unsigned breakpoint:1;
     unsigned :3;
     unsigned user:16;      /* user status */
     unsigned :8;
  } drive;

  struct {
     size_t size;
     unsigned :3;
     unsigned present:1;    /* the drive is present */
     unsigned moving:1;     /* the motor is moving */
     unsigned :3;
     unsigned sequence:1;   /* a sequence is running */
     unsigned :1;
     unsigned error:1;      /* fatal error */
     unsigned trace:1;      /* trace acquisition in progress */
     unsigned :4;
     unsigned :7;
     unsigned warning:1;    /* global warning */
     unsigned :8;
     unsigned :4;
     unsigned breakpoint:1;
     unsigned :3;
     unsigned user:16;      /* user status */
     unsigned :8;
  } dsmax;
  struct {
     size_t size;
     DSA_SW1 sw1;
     DSA_SW2 sw2;
  } sw;
} DSA_STATUS;
```

This structure has a size of 12 bytes. The first 4 bytes represent the size of the structure. It allows the user to modify it without breaking the compatibility with the previous versions. The other 8 bytes represent the bit of the status. Thus, there are 64 bits available for the status.

DSA_STATUS is not a simple structure but an association of several structures. Each one of these structures is peculiar to a type of product. Thus, they allow the user to have access to the same 64 bits in a way specific to a type of products.

The DSMAX structure defines the 64 bits of the DSMAX. The 'drive' structure defines the same 64 bits for a drive. The 'raw' structure allows the user to have access to the 64 bits of the status by two words of 32 bits. The 'sw' structure is done to make the migration easier for the applications which used the previous versions of the DSA library (version 1.0x).

Here is a reading example of the status:

```
int err;
DSA_STATUS status;
...

status.size = 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 user asks for the status of a drive, he has access to the bits of the status via the 'drive' structure.

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

```
DSA_STATUS mask;
DSA_STATUS ref;
...

/* start a movement */
dsa_set_target_movement(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) */
memset(&mask, 0, sizeof(mask));
mask.size = sizeof(mask);
mask.drive.moving = 1;
memset(&ref, 0, sizeof(ref));
ref.size = sizeof(ref);
dsa_wait_status_equal_s(drive1, mask, ref, NULL, 20000);
```

In this example, the user waits for the movement to be finished that is to say that the '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 for this reason that 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 that the user is waiting for are set in the reference. In the example, the value is equal to 0.

In the mask, all the bits except 'moving' must be equal to 0. It is done thanks to the 'memset()' function. 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 absolutely initialized correctly.

It is better to initialize the mask and ref variables at the creation. The simplified code is:

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

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

/* wait for the end of the movement */
mask.drive.moving = 1;
dsa_wait_status_equal_s(drive1, mask, ref, NULL, 20000);
```

In the following example, the user monitors the drive to know if it is in error mode or not. To do so, an asynchronous waiting 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);
}
```

3.11.3 Performances

The status are the fastest way for a device to give Boolean type information to the PC. These information can be a change of state, an event taking place, or any other information. The user has a part of the status to carry the information peculiar to his application.

The routing of these status is then carefully done. The processing at the PC level is realized by interrupt. A task of the PC can be then awakened by a status change. Thus, all the waitings 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

monitorings registers. The status do not have all these information but only the ones which need a fast routing to the PC. For example, there is a bit (number 2) of M60 which indicates if the drive has done the homing or not. This information is really useful but it does not need a routing in real time and then it is not part of the status. If the user wants to know this information, he can simply ask the drive the value of M60 thanks to the dsa_get_register_s() function.

The status are updated every 166 µs on the Turbo-ETEL-Bus (TEB). Then, the DSMAX receives an update of the status at the same rate. Each time the DSMAX detects a change of status, it informs the PC. Thus, the PC is informed of a status change at the same rate than the DSMAX but it is often limited by the performances of the operating system. Only a realtime operating system with a sufficient CPU time will guarantee an update of these status at this rate.

If the performances of the PC do not allow it to monitor the evolution of the status, the temporary changes of these status can be invisible on the PC. 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 could be not noticed by the PC and this bit would be considered as always equal to 1.

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) but simply wait for a given state.

When using the communication via the serial port (EBL, EBL2), the performances of the status are clearly reduced. The update is done every 250 ms. This communication has been mainly developed for the start and the setting. The use of these status with this type of communication is then not interesting.

3.11.4 User status

Among the bits of the status, 16 of them are at the disposal of the user. He can use them as he wishes to carry the information peculiar to his application. At the PC level, these bits are managed exactly like the others. Thus, the user can read them and effectuate waitings on them thanks to the functions described above. At the device level, the user has the K177 parameter which allows him to modify the state of these bits. See the User's Manual of the concerned device to know the functioning of this parameter.

4. Appendixes

4.1 DSA library functions

4.1.1 Functions to send commands

ETEL syntax	Command number	EDI function
ACI	118	
AN1	101	
AN2	102	
ANR	103	
AUT	150	
AXI	109	
BPT	185	
BRK	70	dsa_quick_stop_s()
CAL	68	
CLX	17	
CN1	250	dsa_can_command_1_s()
CN2	251	dsa_can_command_2_s()
CNT	186	
CPE	50	
CPW	51	
CRA	190	
DWN	42	
EDI	62	dsa_edit_sequence_s()
END	0	
ERR	80	
EXI	63	dsa_exit_sequence_s()
FCOS	223	
FFP	225	
FINV	221	
FIP	226	
FSIGN	224	
FSIN	222	
FSQRT	220	
FTST	227	
HLB	121	dsa_quick_stop_s()
HLO	119	dsa_quick_stop_s()
HLT	120	dsa_quick_stop_s()
IBEGIN	553	dsa_ipol_begin_s()
IBRK	653	dsa_ipol_quick_stop_s()

ETEL syntax	Command number	EDI function
ICCW	1041	dsa_ipol_circle_ccw_c2d_s()
ICCWR	1027	dsa_ipol_circle_ccw_r2d_s()
ICLRB	657	
ICONC	1030	dsa_ipol_begin_concatenation_s()
ICONT	654	dsa_ipol_continue_s()
ICW	1040	dsa_ipol_circle_cw_c2d_s()
ICWR	1026	dsa_ipol_circle_cw_r2d_s()
IEND	554	dsa_ipol_end_s()
IEQ	151	
IGE	156	
IGT	154	
ILE	155	
ILINE	1025	dsa_ipol_line_s()
ILINE	1025	dsa_ipol_line_2d_s()
ILOCK	1044	
ILT	153	
IMARK	1039	dsa_ipol_mark_s()
IMROT	1056	
IMSCALE	1055	
IMSHEAR	1057	
IMTRANS	1054	
INCONC	1031	dsa_ipol_end_concatenation_s()
IND	45	dsa_homing_start_s()
INE	152	
IPUSH	1279	
IPVT	1028	dsa_ipol_pvt_s()
IPVT	1028	dsa_ipol_pvt_reg_typ_s()
IPVTUPDATE	662	dsa_ipol_pvt_update_s()
IRST	652	dsa_ipol_reset_s()
ISET	552	dsa_ipol_prepare_s()
ISTP	656	
ITACC	1036	dsa_ipol_tan_acceleration_s()
ITDEC	1037	dsa_ipol_tan_deceleration_s()
ITJRT	1038	dsa_ipol_tan_jerk_time_s()
ITSPD	1035	dsa_ipol_tan_velocity_s()
IUNLOCK	655	
IWTT	1029	
JBC	37	
JBS	36	

ETEL syntax	Command number	EDI function
JEQ	137	
JGT	138	
JLT	136	
JMP	26	dsa_execute_sequence_s()
JNE	139	
LAB	27	
MAS	143	
MSK	40	
MTR	81	
NEW	78	dsa_default_parameters_s()
POP	34	
PSH	255	
PUSH	767	
PWR	124	dsa_power_off_s()
PWR	124	dsa_power_on_s()
PWR	124	dsa_quick_stop_s()
REG	199	
REI	182	
RES	49	dsa_load_parameters_s()
RET	69	
RID	176	
RIE	175	
RSB	87	
RSD	88	
RSH	600	
RST	79	dsa_reset_error_s()
SAV	48	dsa_save_parameters_s()
SET	22	
SLS	46	
STA	25	dsa_change_setpoint_s()
STA	25	dsa_new_setpoint_s()
STE_ABS	129	dsa_step_motion_s()
STE_ADD	114	
STE_SUB	115	
STI	33	
STP	18	dsa_quick_stop_s()
STY	35	
TCL	107	
TMD	180	

ETEL syntax	Command number	EDI function
TMG	181	
TRE	108	
TRI	67	
TRN	106	
TRP	105	
TRR	104	
TST	74	
WAB	13	
WBC	54	
WBS	55	
WPG	53	
WPL	52	
WR04	254	
WR12	252	
WR14	253	
WTB	148	
WTK	513	
WTM	8	dsa_wait_movement_s(), dsa_ipol_wait_movement_s()
WTP	9	dsa_wait_position_s()
WTS	12	
WTT	10	dsa_wait_time_s()
WTW	11	dsa_wait_window_s()
XAC_ADD	161	
XAC_AND	165	
XAC_AND_NOT	167	
XAC_DIV	163	
XAC_MUL	162	
XAC_ORL	164	
XAC_ORL_NOT	166	
XAC_SHL	169	
XAC_SHR	168	
XAC_SUB	160	
XYY_ABS	123	dsa_set_iso_register_s()
XYY_ABS	123	dsa_set_register_s()
XYY_ADD	91	
XYY_AND	95	
XYY_AND_NOT	97	
XYY_CONV	122	
XYY_DIV	94	

ETEL syntax	Command number	EDI function
XYY_MUL	93	
XYY_NOT	174	
XYY_ORL	96	
XYY_ORL_NOT	98	
XYY_SHL	173	
XYY_SHR	172	
XYY_SUB	92	
ZCL	211	dsa_set_trace_mode_pos_s()
ZCL	211	dsa_set_trace_mode_dev_s()
ZCL	211	dsa_set_trace_mode_iso_s()
ZFT	203	
ZIM	210	dsa_set_trace_mode_pos_s()
ZIM	210	dsa_set_trace_mode_mvt_s()
ZIM	210	dsa_set_trace_mode_iso_s()
ZIM	210	dsa_set_trace_mode_dev_s()
ZIM	210	dsa_set_trace_mode_immediate_s()
ZTE	204	dsa_sync_trace_enable_s()
ZTE	204	dsa_sync_trace_force_trigger_s()
ZTR	202	dsa_trace_acquisition_s()

4.1.2 Functions for the reading and the writing of the registers

Registers	Alias	EDI functions	
K1		dsa_get_pl_proportional_gain_s()	dsa_set_pl_proportional_gain_s()
K2		dsa_get_pl_speed_feedback_gain_s()	dsa_set_pl_speed_feedback_gain_s()
K3		dsa_get_pl_force_feedback_gain_1_s()	dsa_set_pl_force_feedback_gain_1_s()
K4		dsa_get_pl_integrator_gain_s()	dsa_set_pl_integrator_gain_s()
K5		dsa_get_pl_anti_windup_gain_s()	dsa_set_pl_anti_windup_gain_s()
K6		dsa_get_pl_integrator_limitation_s()	dsa_set_pl_integrator_limitation_s()
K7		dsa_get_pl_integrator_mode_s()	dsa_set_pl_integrator_mode_s()
K8		dsa_get_pl_speed_filter_s()	dsa_set_pl_speed_filter_s()
K9		dsa_get_pl_output_filter_s()	dsa_set_pl_output_filter_s()
K10		dsa_get_cl_input_filter_s()	dsa_set_cl_input_filter_s()
K11		dsa_get_ttl_special_filter_s()	dsa_set_ttl_special_filter_s()
K13		dsa_get_pl_force_feedback_gain_2_s()	dsa_set_pl_force_feedback_gain_2_s()
K20		dsa_get_pl_speed_feedfwd_gain_s()	dsa_set_pl_speed_feedfwd_gain_s()
K21		dsa_get_pl_acc_feedforward_gain_s()	dsa_set_pl_acc_feedforward_gain_s()
K23		dsa_get_cl_phase_advance_factor_s()	dsa_set_cl_phase_advance_factor_s()
K24		dsa_get_apr_input_filter_s()	dsa_set_apr_input_filter_s()
K25		dsa_get_cl_phase_advance_shift_s()	dsa_set_cl_phase_advance_shift_s()
K26		dsa_get_min_position_range_limit_s()	dsa_set_min_position_range_limit_s()
K27		dsa_get_max_position_range_limit_s()	dsa_set_max_position_range_limit_s()
K28		dsa_get_max_profile_velocity_s()	dsa_set_max_profile_velocity_s()
K29		dsa_get_max_acceleration_s()	dsa_set_max_acceleration_s()
K30		dsa_get_following_error_window_s()	dsa_set_following_error_window_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()
K33		dsa_get_enable_input_mode_s()	dsa_set_enable_input_mode_s()
K34		dsa_get_min_soft_position_limit_s()	dsa_set_min_soft_position_limit_s()
K35		dsa_get_max_soft_position_limit_s()	dsa_set_max_soft_position_limit_s()
K36		dsa_get_profile_limit_mode_s()	dsa_set_profile_limit_mode_s()
K37		dsa_get_io_error_event_mask_s()	dsa_set_io_error_event_mask_s()
K38		dsa_get_position_window_time_s()	dsa_set_position_window_time_s()
K39		dsa_get_position_window_s()	dsa_set_position_window_s()
K40		dsa_get_homing_method_s()	dsa_set_homing_method_s()
K41		dsa_get_homing_zero_speed_s()	dsa_set_homing_zero_speed_s()
K42		dsa_get_homing_acceleration_s()	dsa_set_homing_acceleration_s()
K43		dsa_get_homing_following_limit_s()	dsa_set_homing_following_limit_s()
K44		dsa_get_homing_current_limit_s()	dsa_set_homing_current_limit_s()
K45		dsa_get_home_offset_s()	dsa_set_home_offset_s()
K46		dsa_get_homing_fixed_mvt_s()	dsa_set_homing_fixed_mvt_s()

Registers	Alias	EDI functions	
K47		dsa_get_homing_switch_mvt_s()	dsa_set_homing_switch_mvt_s()
K48		dsa_get_homing_index_mvt_s()	dsa_set_homing_index_mvt_s()
K50		dsa_get_profile_mul_shift_s()	dsa_set_profile_mul_shift_s()
K51		dsa_get_encoder_motor_ratio_s()	dsa_set_encoder_motor_ratio_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()
K54		dsa_get_motor_pole_nb_s()	dsa_set_motor_pole_nb_s()
K55		dsa_get_encoder_turn_factor_s()	dsa_set_encoder_turn_factor_s()
K56		dsa_get_motor_phase_correction_s()	dsa_set_motor_phase_correction_s()
K60		dsa_get_software_current_limit_s()	dsa_set_software_current_limit_s()
K61		dsa_get_drive_control_mode_s()	dsa_set_drive_control_mode_s()
K63		dsa_get_drive_fast_seq_mode_s()	dsa_set_drive_fast_seq_mode_s()
K64		dsa_get_drive_sp_factor_s()	dsa_set_drive_sp_factor_s()
K66		dsa_get_display_mode_s()	dsa_set_display_mode_s()
K68		dsa_get_encoder_inversion_s()	dsa_set_encoder_inversion_s()
K69		dsa_get_pdr_step_value_s()	dsa_set_pdr_step_value_s()
K70		dsa_get_encoder_phase_1_offset_s()	dsa_set_encoder_phase_1_offset_s()
K71		dsa_get_encoder_phase_2_offset_s()	dsa_set_encoder_phase_2_offset_s()
K72		dsa_get_encoder_phase_1_factor_s()	dsa_set_encoder_phase_1_factor_s()
K73		dsa_get_encoder_phase_2_factor_s()	dsa_set_encoder_phase_2_factor_s()
K74		dsa_get_encoder_phase_3_offset_s()	dsa_set_encoder_phase_3_offset_s()
K75		dsa_get_encoder_index_distance_s()	dsa_set_encoder_index_distance_s()
K76		dsa_get_encoder_phase_3_factor_s()	dsa_set_encoder_phase_3_factor_s()
K77		dsa_get_encoder_ipol_shift_s()	dsa_set_encoder_ipol_shift_s()
K78		dsa_get_encoder_mul_shift_s()	dsa_set_encoder_mul_shift_s()
K79		dsa_get_encoder_type_s()	dsa_set_encoder_type_s()
K80		dsa_get_cl_proportional_gain_s()	dsa_set_cl_proportional_gain_s()
K81		dsa_get_cl_integrator_gain_s()	dsa_set_cl_integrator_gain_s()
K82		dsa_get_cl_output_filter_s()	dsa_set_cl_output_filter_s()
K83		dsa_get_cl_current_limit_s()	dsa_set_cl_current_limit_s()
K84		dsa_get_cl_i2t_current_limit_s()	dsa_set_cl_i2t_current_limit_s()
K85		dsa_get_cl_i2t_time_limit_s()	dsa_set_cl_i2t_time_limit_s()
K86		dsa_get_cl_regen_mode_s()	dsa_set_cl_regen_mode_s()
K86		dsa_get_encoder_hall_phase_adj_s()	dsa_set_encoder_hall_phase_adj_s()
K89		dsa_get_motor_phase_nb_s()	dsa_set_motor_phase_nb_s()
K90		dsa_get_init_mode_s()	dsa_set_init_mode_s()
K91		dsa_get_init_pulse_level_s()	dsa_set_init_pulse_level_s()
K92		dsa_get_init_max_current_s()	dsa_set_init_max_current_s()
K93		dsa_get_init_final_phase_s()	dsa_set_init_final_phase_s()

Registers	Alias	EDI functions	
K94		dsa_get_init_time_s()	dsa_set_init_time_s()
K95		dsa_get_init_current_rate_s()	dsa_set_init_current_rate_s()
K96		dsa_get_init_phase_rate_s()	dsa_set_init_phase_rate_s()
K97		dsa_get_init_initial_phase_s()	dsa_set_init_initial_phase_s()
K140		dsa_get_drive_fuse_checking_s()	dsa_set_drive_fuse_checking_s()
K141		dsa_get_motor_temp_checking_s()	dsa_set_motor_temp_checking_s()
K150	MST	dsa_get_mon_source_type_s()	dsa_set_mon_source_type_s()
K151	MSN	dsa_get_mon_source_index_s()	dsa_set_mon_source_index_s()
K153	MDN	dsa_get_mon_dest_index_s()	dsa_set_mon_dest_index_s()
K154	MOF	dsa_get_mon_offset_s()	dsa_set_mon_offset_s()
K155	MAM	dsa_get_mon_gain_s()	dsa_set_mon_gain_s()
K156	DAO	dsa_get_x_analog_offset_s()	dsa_set_x_analog_offset_s()
K157	DAA	dsa_get_x_analog_gain_s()	dsa_set_x_analog_gain_s()
K160		dsa_get_syncro_input_mask_s()	dsa_set_syncro_input_mask_s()
K161		dsa_get_syncro_input_value_s()	dsa_set_syncro_input_value_s()
K162		dsa_get_syncro_output_mask_s()	dsa_set_syncro_output_mask_s()
K163		dsa_get_syncro_output_value_s()	dsa_set_syncro_output_value_s()
K164		dsa_get_syncro_start_timeout_s()	dsa_set_syncro_start_timeout_s()
K170	MDE		
K171	DOUT	dsa_get_digital_output_s()	dsa_set_digital_output_s()
K172		dsa_get_x_digital_output_s()	dsa_set_x_digital_output_s()
K173		dsa_get_x_analog_output_1_s()	dsa_set_x_analog_output_1_s()
K174		dsa_get_x_analog_output_2_s()	dsa_set_x_analog_output_2_s()
K175		dsa_get_analog_output_s()	dsa_set_analog_output_s()
K180		dsa_get_interrupt_mask_1_s()	dsa_set_interrupt_mask_1_s()
K181		dsa_get_interrupt_mask_2_s()	dsa_set_interrupt_mask_2_s()
K184		dsa_get_trigger_irq_mask_s()	dsa_set_trigger_irq_mask_s()
K185	TMK	dsa_get_trigger_io_mask_s()	dsa_set_trigger_io_mask_s()
K186	TRS	dsa_get_trigger_map_offset_s()	dsa_set_trigger_map_offset_s()
K187	TNB	dsa_get_trigger_map_size_s()	dsa_set_trigger_map_size_s()
K190		dsa_get_realtime_enabled_global_s()	dsa_set_realtime_enabled_global_s()
K191		dsa_get_realtime_valid_mask_s()	dsa_set_realtime_valid_mask_s()
K192		dsa_get_realtime_enabled_mask_s()	dsa_set_realtime_enabled_mask_s()
K193		dsa_get_realtime_pending_mask_s()	dsa_set_realtime_pending_mask_s()
K195		dsa_get_ebl_baudrate_s()	dsa_set_ebl_baudrate_s()
K197		dsa_get_indirect_axis_number_s()	dsa_set_indirect_axis_number_s()
K198		dsa_get_indirect_register_idx_s()	dsa_set_indirect_register_idx_s()
K199		dsa_get_indirect_register_sidx_s()	dsa_set_indirect_register_sidx_s()
K201	MMC	dsa_get_concatenated_mvt_s()	dsa_set_concatenated_mvt_s()

Registers	Alias	EDI functions	
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()
K205	CAM	dsa_get_came_value_s()	dsa_set_came_value_s()
K206		dsa_get_brake_deceleration_s()	dsa_set_brake_deceleration_s()
K210	POS	dsa_get_target_position_s()	dsa_set_target_position_s()
K211	SPD	dsa_get_profile_velocity_s()	dsa_set_profile_velocity_s()
K212	ACC	dsa_get_profile_acceleration_s()	dsa_set_profile_acceleration_s()
K213	JRT	dsa_get_jerk_filter_time_s()	dsa_set_jerk_filter_time_s()
K214		dsa_get_profile_deceleration_s()	dsa_set_profile_deceleration_s()
K215		dsa_get_end_velocity_s()	dsa_set_end_velocity_s()
K220		dsa_get_ctrl_source_type_s()	dsa_set_ctrl_source_type_s()
K221		dsa_get_ctrl_source_index_s()	dsa_set_ctrl_source_index_s()
K222		dsa_get_ctrl_shift_factor_s()	dsa_set_ctrl_shift_factor_s()
K223		dsa_get_ctrl_offset_s()	dsa_set_ctrl_offset_s()
K224		dsa_get_ctrl_gain_s()	dsa_set_ctrl_gain_s()
K239		dsa_get_motor_kt_factor_s()	dsa_set_motor_kt_factor_s()
K240		dsa_get_motor_type_s()	dsa_set_motor_type_s()
K241		dsa_get_encoder_period_s()	dsa_set_encoder_period_s()
K242		dsa_get_motor_mul_factor_s()	dsa_set_motor_mul_factor_s()
K243		dsa_get_motor_div_factor_s()	dsa_set_motor_div_factor_s()
K244	NAME		
K530		dsa_ipol_set_velocity_rate_s()	
M2		dsa_get_position_ctrl_error_s()	
M3		dsa_get_position_max_error_s()	
M6		dsa_get_position_demand_value_us_s()	
M7		dsa_get_position_actual_value_us_s()	
M10		dsa_get_velocity_demand_value_s()	
M11		dsa_get_velocity_actual_value_s()	
M14		dsa_get_acc_demand_value_s()	
M15		dsa_get_acc_actual_value_s()	
M18		dsa_get_ref_demand_value_s()	
M19		dsa_get_drive_control_mode_bf_s()	
M20		dsa_get_cl_current_phase_1_s()	
M21		dsa_get_cl_current_phase_2_s()	
M22		dsa_get_cl_current_phase_3_s()	
M25		dsa_get_cl_lkt_phase_1_s()	
M26		dsa_get_cl_lkt_phase_2_s()	
M27		dsa_get_cl_lkt_phase_3_s()	

Registers	Alias	EDI functions	
M30		dsa_get_cl_demand_value_s()	
M31		dsa_get_cl_actual_value_s()	
M40		dsa_get_encoder_sine_signal_s()	
M41		dsa_get_encoder_cosine_signal_s()	
M42		dsa_get_encoder_index_signal_s()	
M45		dsa_get_encoder_hall_1_signal_s()	
M46		dsa_get_encoder_hall_2_signal_s()	
M47		dsa_get_encoder_hall_3_signal_s()	
M48		dsa_get_encoder_hall_dig_signal_s()	
M50	DIN	dsa_get_digital_input_s()	
M51	AI1	dsa_get_analog_input_s()	
M55		dsa_get_x_digital_input_s()	
M56		dsa_get_x_analog_input_1_s()	
M57		dsa_get_x_analog_input_2_s()	
M60	SD1	dsa_get_drive_status_1_s()	
M61	SD2	dsa_get_drive_status_2_s()	
M64		dsa_get_error_code_s()	
M67		dsa_get_cl_i2t_value_s()	
M70		dsa_get_info_product_number_s()	
M70		dsa_get_info()	
M71		dsa_get_info_boot_revision_s()	
M71		dsa_get_info()	
M72	VER	dsa_get_info_soft_version_s()	
M72	VER	dsa_get_info()	
M73	SER	dsa_get_info_serial_number_s()	
M73	SER	dsa_get_info()	
M74		dsa_get_info_c_soft_build_time_s()	
M74		dsa_get_info()	
M75		dsa_get_info_p_soft_build_time_s()	
M75		dsa_get_info()	
M76		dsa_get_x_info_product_number_s()	
M76		dsa_get_x_info()	
M77		dsa_get_x_info_boot_revision_s()	
M77		dsa_get_x_info()	
M78		dsa_get_x_info_soft_version_s()	
M78		dsa_get_x_info()	
M79		dsa_get_x_info_serial_number_s()	
M79		dsa_get_x_info()	
M80		dsa_get_x_info_soft_build_time_s()	

Registers	Alias	EDI functions	
M80		dsa_get_x_info()	
M82		dsa_get_drive_max_current_s()	
M85		dsa_get_info_product_string_s()	
M85		dsa_get_info()	
M86		dsa_get_x_info_product_string_s()	
M86		dsa_get_x_info()	
M87		dsa_get_axis_number_s()	
M88		dsa_get_daisy_chain_number_s()	
M90		dsa_get_drive_temperature_s()	
M93		dsa_get_drive_mask_value_s()	
M95		dsa_get_drive_display_s()	
M96		dsa_get_drive_sequence_line_s()	
M140		dsa_get_drive_fuse_status_s()	
M160		dsa_get_irq_drive_status_1_s()	
M161		dsa_get_irq_drive_status_2_s()	
M162		dsa_get_ack_drive_status_1_s()	
M163		dsa_get_ack_drive_status_2_s()	
M164		dsa_get_irq_pending_axis_mask_s()	
M241		dsa_get_encoder_ipol_factor_s()	
M242		dsa_get_drive_quartz_frequency_s()	
M243		dsa_get_drive_cl_time_factor_s()	
M244		dsa_get_drive_pl_time_factor_s()	
M245		dsa_get_drive_sp_time_factor_s()	
M250		dsa_get_can_feedback_1_s()	
M251		dsa_get_can_feedback_2_s()	

4.2 Prototypes of generic functions

4.2.1 Types

```
typedef struct DsaCommandParam {
       int typ;             /* parameter type */
       int conv;            /* conversion ID, zero means no conversion */
       union {
              int i;        /* used when conv = 0 */
              double d;     /* used when conv != 0 */
       } val;
} DSA_COMMAND_PARAM;
```

4.2.2 Functions to send commands

int **dsa_execute_command_s**(DSA_DEVICE_BASE *grp, int cmd,
bool fast, bool ereport, long timeout);

int **dsa_execute_command_d_s**(DSA_DEVICE_BASE *grp, int cmd,
int typ1, long par1, bool fast, bool ereport, long timeout);

int **dsa_execute_command_i_s**(DSA_DEVICE_BASE *grp, int cmd,
int typ1, double par1, int conv1, bool fast, bool ereport, long timeout);

int **dsa_execute_command_dd_s**(DSA_DEVICE_BASE *grp, int cmd,
int typ1, long par1, int typ2, long par2,
bool fast, bool ereport, long timeout);

int **dsa_execute_command_id_s**(DSA_DEVICE_BASE *grp, int cmd,
int typ1, double par1, int conv1, int typ2, long par2,
bool fast, bool ereport, long timeout);

int **dsa_execute_command_di_s**(DSA_DEVICE_BASE *grp, int cmd,
int typ1, long par1, int typ2, double par2, int conv2,
bool fast, bool ereport, long timeout);

int **dsa_execute_command_ii_s**(DSA_DEVICE_BASE *grp, int cmd,
int typ1, double par1, int conv1, int typ2, double par2, int conv2,
bool fast, bool ereport, long timeout);

int **dsa_execute_command_x_s**(DSA_DEVICE_BASE *grp, int cmd,
DSA_COMMAND_PARAM *params, int count,
bool fast, bool ereport, long timeout);

int **dsa_execute_command_a**(DSA_DEVICE_BASE *grp, int cmd,
bool fast, bool ereport, DSA_HANDLER handler, void *param);

int **dsa_execute_command_d_a**(DSA_DEVICE_BASE *grp, int cmd,
int typ1, long par1, bool fast, bool ereport, DSA_HANDLER handler, void *param);

int **dsa_execute_command_i_a**(DSA_DEVICE_BASE *grp, int cmd,
int typ1, double par1, int conv1,
bool fast, bool ereport, DSA_HANDLER handler, void *param);

int **dsa_execute_command_dd_a**(DSA_DEVICE_BASE *grp, int cmd,
int typ1, long par1, int typ2, long par2,
bool fast, bool ereport, DSA_HANDLER handler, void *param);

int **dsa_execute_command_id_a**(DSA_DEVICE_BASE *grp, int cmd,

int typ1, double par1, int conv1, int typ2, long par2,
bool fast, bool ereport, DSA_HANDLER handler, void *param);

int **dsa_execute_command_di_a**(DSA_DEVICE_BASE *grp, int cmd,
int typ1, long par1, int typ2, double par2, int conv2,
bool fast, bool ereport, DSA_HANDLER handler, void *param);

int **dsa_execute_command_ii_a**(DSA_DEVICE_BASE *grp, int cmd,
int typ1, double par1, int conv1, int typ2, double par2, int conv2,
bool fast, bool ereport, DSA_HANDLER handler, void *param);

int **dsa_execute_command_x_a**(DSA_DEVICE_BASE *grp, int cmd,
DSA_COMMAND_PARAM *params, int count,
bool fast, bool ereport, DSA_HANDLER handler, void *param);

4.2.3 Functions for the reading and writing of registers

int **dsa_get_register_s**(DSA_DEVICE *dev,
int typ, unsigned idx, int sidx, long *val,
int kind, long timeout);

int **dsa_set_register_s**(DSA_DEVICE_BASE *grp,
int typ, unsigned idx, int sidx, long val, long timeout);

int **dsa_get_iso_register_s**(DSA_DEVICE *dev,
int typ, unsigned idx, int sidx, double *val, int conv, int kind, long timeout);

int **dsa_set_iso_register_s**(DSA_DEVICE_BASE *grp,
int typ, unsigned idx, int sidx, double val, int conv, long timeout);

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

int **dsa_set_register_a**(DSA_DEVICE_BASE *grp,
int typ, unsigned idx, int sidx, long val, DSA_HANDLER handler, void *param);

int **dsa_get_iso_register_a**(DSA_DEVICE *dev,
int typ, unsigned idx, int sidx, int conv, int kind,
DSA_DOUBLE_HANDLER handler, void *param);

int **dsa_set_iso_register_a**(DSA_DEVICE_BASE *grp,
int typ, unsigned idx, int sidx, double val, int conv,
DSA_HANDLER handler, void *param);

4.2.4 Functions for the management of the status

int **dsa_get_status**(DSA_DEVICE *drv, DSA_STATUS *status);

int **dsa_wait_status_equal_s**(DSA_DEVICE *drv,
DSA_STATUS *mask, DSA_STATUS *ref, DSA_STATUS *status, long timeout);

int **dsa_wait_status_not_equal_s**(DSA_DEVICE *drv,
DSA_STATUS *mask, DSA_STATUS *ref, DSA_STATUS *status, long timeout);

int **dsa_grp_wait_and_status_equal_s**(DSA_DEVICE_GROUP *grp,
DSA_STATUS *mask, DSA_STATUS *ref, long timeout);

int **dsa_grp_wait_and_status_not_equal_s**(DSA_DEVICE_GROUP *grp,
DSA_STATUS *mask, DSA_STATUS *ref, long timeout);

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

int **dsa_wait_status_not_equal_a**(DSA_DEVICE *drv,
DSA_STATUS *mask, DSA_STATUS *ref,
DSA_STATUS_HANDLER handler, void *param);

int **dsa_grp_wait_and_status_equal_a**(DSA_DEVICE_GROUP *grp,
DSA_STATUS *mask, DSA_STATUS *ref,
DSA_HANDLER handler, void *param);

int **dsa_grp_wait_and_status_not_equal_a**(DSA_DEVICE_GROUP *grp,
DSA_STATUS *mask, DSA_STATUS *ref,
DSA_HANDLER handler, void *param);

int **dsa_cancel_status_wait**(DSA_DEVICE_BASE *grp);

4.3 Example in C

4.3.1 Examples in C of a communication with a drive

First example:

```
/*
 * sample_1.c 1.01a
 *
 * Copyright (c) 2001 ETEL SA. All rights reserved.
 *
 * This software is confidential and proprietary information of ETEL SA
 * ("Confidential Information"). You shall not disclose such Confidential
 * Information and shall use it only in accordance with the terms of the
 * license agreement you entered into with ETEL.
 *
 * This software is provided "AS IS," without a warranty 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.
 *
 * This 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. Licensee represents and warrants that it will not use or
 * redistribute the software for such purposes.
 */

/*
 * This simple demo program show 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 ETEL position loop controller properly configured
 *  - an ETEL-Bus-Lite or ETEL-Bus-Lite2 connection between the drive and
 *  the PC on COM1.
 *  - the drive must have the axis number 1.
 *  - k45 must be set to insure that position 0 is inside the valid range.
 *  - k34 and k35 must be set properly.
 * If k34 and k35 aren't set properly, you can modify this code to set
 * pos_min and pos_max manually.
 */

/*** libraries ***/

/*
 * Include the standard I/O library.
 */
```

```
#include <stdio.h>

/*
 * Include EDI high level library version 2.0x.
 */
#include <dsa20.h>

/*** functions ***/

/*
 * Entry point function.
 */
int main(char argc, int *argv[])
{
    /*
     * We need two variables to hold the minimum and maximum position of the
     * motor. All parameters which represent physical dimensions are usually
     * defined in ISO units and use double precision floating point variables
     */
    double pos_min, pos_max;

    /*
     * We must also define a pointer to a DSA_DRIVE object. This hidden
     * object is defined in the dsa20 library. The client does not need to
     * access the members of this object directly, but need to pass it to
     * various dsa library functions. It can be compared to the FILE
     * structure defined in the standard I/O library.
     * The drv pointer *must* be initialized to NULL before calling the
     * created drive function, otherwise this function will fail.
     */
    DSA_DRIVE *drv = NULL;

    /*
     * Most functions of the library return an error code or 0 if no error
     * occured. We need a variable to store the last error.
     * The error codes are negative values, ranging from -399 to -300 for
     * the dsa20 library.
     */
    int err;

    /*
     * The first operation is to create the drive object. This operation
     * just allocates some memory and OS resources and initializes it. No
     * active communication takes place at this step.
     * The address of the drive object is returned into the pointer passed by
     * reference to the function.
     * If an error occurs, we jump to the error label. It is usually not very
     * nice to use the goto keyword, but it's sometimes the best way for
     * error handling in C.
     */
    err = dsa_create_drive(&drv);
    if(err != 0)
        goto _error;

    /*
     * The next thing to do is to open the communication with the drive. This
     * is usually done by the dsa_open_u function, which takes the url of the
     * drive as the parameter. At this time, only ETEL-Bus's URLs are
     * recognized.
     * It must take the following form: "etb:<driver>:<axis>"
     * The driver part of the url is the same string that is passed to the
```

```
 * ETEL-Bus etb_open low level function. Valid driver string includes:
 *  - "COM<number>[:baudrate]"
 *      for direct com port (serial port) access.
 *       - number: the com port number, usually 1 or 2.
 *       - baudrate: communication speed. default is 115200 bauds
 *  - "ETN://<host>:<port>"
 *      for local or remote access via the TCP/IP protocol to the ETEL
 *      network daemon (etnd), DSMAX2, ...
 *       - host is the DNS name of the computer, or it's IP address.
 *         (localhost or 127.0.0.1 usually represent the local computer)
 *       - port is the TCP port number.
 *  - "TCP://<host>:<port>"
 *      actually replaced ba the ETN driver
 *  - "DUMMY:<amsk>:<d_prod>:<d_ver>[:x_prod[:x_ver]]"
 *      for drive "simulation" with
 *       - amsk:bit field of all axes present on the bus(5 means 0 and 2)
 *       - d_prod: drive product number (2 = DSA2P, 4 = DSB2P, see M70)
 *       - d_ver: drive software version (see M72)
 *       - x_prod: extension card product number (see M76)
 *       - x_ver: extension card software version (see M78)
 * This function is likely to fail if the drive is not running, has a bad
 * connection, if the port is already in use, ...
 */
err = dsa_open_u(drv, "etb:COM1:1");
if(err != 0)
    goto _error;

/*
 * Now we can send commands to the drive. The first thing to do is to
 * put the drive in power on state. This is done by the dsa_power_on_s
 * command. The "_s" at the end of the command indicates that this is a
 * *synchronous* function. Synchronous function are waiting 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 drive before the end of the specified timeout. This
 * lack of response usually indicates an error in the application, or
 * could result from bad drive parameters. An appropriate timeout value
 * heavily depends on the application. In the power on case, less than 1
 * second could be appropriate with pulse initialization, but more than 5
 * seconds could be required with constant current initialization.
 */
err = dsa_power_on_s(drv, 10000);
if(err != 0)
    goto _error;

/*
 * An indexation command must then be issued in order to find the
 * absolute position of the motor. The "homing start" command will just
 * *start* the homing procedure.
 */
err = dsa_homing_start_s(drv, 10000);
if(err != 0)
    goto _error;

/*
 * During the indexation procedure, we can read the minimum and maximum
 * position allowed for the motor. This is done by calling the function
 * "get min/max soft position limit". This function writes the data at
 * the address provided in the second argument. The third argument
 * indicates to the function what kind of value must be returned: the
```

```
 * current value, the default value, the maximum or minimum value,...
 * Some of these operations (getting min, max or default values) do not
 * involve any communication.
 * As usually, a timeout must be provided, but for request operation,
 * which are always executed in one cycle in the drive, it is recommended
 * to ask for the default timeout. This special timeout value is
 * automatically adjusted to the communication channel used.
 */
err = dsa_get_min_soft_position_limit_s(drv, &pos_min, DSA_GET_CURRENT,
DSA_DEF_TIMEOUT);
if(err != 0)
    goto _error;
err = dsa_get_max_soft_position_limit_s(drv, &pos_max, DSA_GET_CURRENT,
DSA_DEF_TIMEOUT);
if(err != 0)
    goto _error;

/*
 * Now we will wait for the end of the indexation procedure before
 * continuing this program. This is done with the "wait movement" command
 * The only required parameter is the timeout, and its correct value
 * is application dependent. A conservative value of 1 minute is given
 * here to accommodate most situations.
 */
err = dsa_wait_movement_s(drv, 60000);
if(err != 0)
    goto _error;

/*
 * The next command will start a movement near (95%) the first limit.
 * This is done with the "set target position" function, which is the
 * equivalent of the standard POS function in the drive. The function
 * takes a subindex parameter as it's second argument. If a zero is
 * written in the subindex, 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.
 * The "set target position" command will just *start* the movement,
 * which means that the operation will just need one or two milliseconds
 * (on the ETEL-Bus-Lite) to be executed by the drive, and then the
 * response will come back quickly to the master. In that case, the
 * timeout value that need to be supplied must only take the
 * communication time from the pc to the drive into account. In this
 * situation, a special value can be used, DSA_DEF_TIMEOUT, which ask the
 * function to use a default timeout value adjusted to the communication
 * channel used. After the command, we will again wait for the end of
 * the motion. The timeout value has been fixed to 1 minute for each of
 * the following movement waiting functions.
 */
err = dsa_set_target_position_s(drv, 0, pos_min * 0.95 + pos_max * 0.05,
DSA_DEF_TIMEOUT);
if(err != 0)
    goto _error;
err = dsa_wait_movement_s(drv, 60000);
if(err != 0)
    goto _error;

/*
 * Now we move to the other limit, followed by a move to the zero
 * position, before putting the motor in power off state.
 */
/*
```

```
         * Move to 95% of the second limit.
         */
        err = dsa_set_target_position_s(drv, 0, pos_min * 0.05 + pos_max * 0.95,
        DSA_DEF_TIMEOUT);
        if(err != 0)
            goto _error;
        err = dsa_wait_movement_s(drv, 60000);
        if(err != 0)
            goto _error;

        /*
         * Move to 95% of the second limit.
         */
        err = dsa_set_target_position_s(drv, 0, pos_min * 0.05 + pos_max * 0.95,
        DSA_DEF_TIMEOUT);
        if(err != 0)
            goto _error;
        err = dsa_wait_movement_s(drv, 60000);
        if(err != 0)
            goto _error;

        /*
         * Power off the motor.
         */
        err = dsa_power_off_s(drv, 60000);
        if(err != 0)
            goto _error;

        /*
         * The first thing to do in order to cleanly terminate the program is to
         * close the communication to the drive.
         */
        err = dsa_close(drv);
        if(err != 0)
            goto _error;

        /*
         * The last operation is to release the memory and resources used by the
         * drive object. In this case, this operation is not required because we
         * are at the end of the program and the OS will free theses resources
         * itself as soon as the program terminates. After freeing the resources,
         * the function will reassign NULL to the drive pointer.
         */
        err = dsa_destroy(&drv);
        if(err != 0)
            goto _error;

        /*
         * Everything seems successful, we can exit now.
         */
        fprintf(stderr, "success.\n");
        return 0;

        /*
         * Now comes the error handler. Because we want to implement only one
         * common error handler, we will have to check the state of the program
         * to know how to terminate the program in the best way.
         */

    _error:
        /*
```

```
         * The first test checks if the drv pointer itself is valid.
         * An invalid pointer would mean that the dsa_create_drive() function
         * has failed or that memory corruption occured.
         */
        if(dsa_is_valid_drive(drv)) {

            /*
             * Now we will check if the connection has been established. A closed
             * connection would indicate that dsa_open_u() failed.
             * We don't care about the returned value of this function because we
             * are only interested in the original error that occured. However,
             * the open flag has to be initialized to false to properly handle
             * the possibility of a failure in this function.
             */
            bool open = 0;
            dsa_is_open(drv, &open);
            if(open) {

                /*
                 * Now it's time to stop the motor. the dsa_quick_stop_s()
                 * function will bypass any other command pending into the drive.
                 * If the motor is already stopped, this operation doesn't have
                 * any effect.
                 */
                dsa_quick_stop_s(drv, 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(drv, 60000);
                dsa_power_off_s(drv, 60000);

                /*
                 * At this point, we can close the communication.
                 */
                dsa_close(drv);
            }

            /*
             * And finally, we release all resources to the OS.
             */
            dsa_destroy(&drv);
        }

        /*
         * Now we will print a small message describing the error which occured.
         * The "translate error" function returns a short text description of the
         * given error. The main function will return with return code 1 to
         * indicate an error.
         */
        fprintf(stderr, "error %d: %s.\n", err, dsa_translate_error(err));
        return 1;
    }
```

Second example:

```
/*
 * sample_2.c 1.01a
 *
 * Copyright (c) 2001 ETEL SA. All rights reserved.
 *
 * This software is confidential and proprietary information of ETEL SA
 * ("Confidential Information"). You shall not disclose such Confidential
 * Information and shall use it only in accordance with the terms of the
 * license agreement you entered into with ETEL.
 *
 * This software is provided "AS IS," without a warranty 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.
 *
 * This 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. Licensee represents and warrants that it will not use or
 * redistribute the software for such purposes.
 */

/*
 * 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 motor with two ETEL position loop controllers configured.
 *  - an ETEL-Bus-Lite or ETEL-Bus_Lite2 connection between the drive and
 *    the PC on COM1.
 *  - an ETEL-Bus or Turbo-ETEL-Bus connection between the two ETEL
 *    controllers
 *  - the drives must have the axes number 0 (for the master) and 1 (slave).
 *  - k45 must be set to insure that position 0 is inside the valid range.
 *  - k34 and k35 must be set properly.
 */

/*** libraries ***/

/*
 * Include the standard I/O library and utility library (for rand()).
 */
#include <stdio.h>
#include <stdlib.h>

/*
 * Include EDI high level library version 2.0x.
 */
#include <dsa20.h>
```

```
/*** prototypes ***/
/*
 * This utility function is defined to perform a simple xy movement.
 */
static int move_xy(DSA_DRIVE_GROUP* xy, double x, double y);

/*** functions ***/

/*
 * Entry point function.
 */
int main(char argc, int *argv[])
{
    /* Stores the min/max position of each axis. */
    double pos_min[2], pos_max[2];

    /*
     * Now, we must define two pointers on DSA_DRIVE objects, for each axis.
     * We must also define a pointer to a DSA_DRIVE_GROUP object. This hidden
     * object is also defined in the dsa20 library, and can be considered as
     * a sort of array containing other DSA_DRIVE objects. The client can
     * set, change and retrieve the different drives belonging to this object
     * through some accessor functions. As with DSA_DRIVE, the drv pointer
     * *must* be initialized to NULL before calling the created group
     * function, otherwise this function will fail.
     */
    DSA_DRIVE *drv[2] = { NULL, NULL };
    DSA_DRIVE_GROUP *grp = NULL;

    /* Defines some global variables. */
    int err;
    int i;

    /* Create two drive objects. */
    for(i = 0; i < 2; i++) {
        err = dsa_create_drive(drv + i);
        if(err < 0)
            goto _error;
    }

    /*
     * Now, we should create the groupe object. The size for the group must
     * be given at the creation time, and cannot currently be changed
     * afterwards. In our case, we want to put two drives in this group.
     */
    err = dsa_create_drive_group(&grp, 2);
    if(err < 0)
        goto _error;

    /*
     * After creating the group, it should be filled with the required
     * drives. The second argument in the dsa_set_group_item() function is
     * the position inside the array where the drive object must be set.
     * In place of dsa_set_group_item(), you can call dsa_add_group_item()
     * that need only two arguments: the group and the drive to put on in.
     */
    for(i = 0; i < 2; i++) {
        err = dsa_set_group_item(grp, i, drv[i]);
        if(err < 0)
            goto _error;
    }
```

```
/*
 * Now, we must establish the connection by calling the dsa_open_u()
 * function for each drive, as previously done. Note that the first call
 * in the dsa_open_u() function for a specific bus will take more time
 * to be performed than subsequent ones, because the underlaying ETB_BUS
 * object used to access the drives is shared by all DSA_DRIVE objects
 * and the first call to dsa_open_u() will have to initialize and open it
 */
err = dsa_open_u(drv[0], "etb:COM1:0");
if(err < 0)
    goto _error;
err = dsa_open_u(drv[1], "etb:COM1:1");
if(err < 0)
    goto _error;

/*
 * Now we can send commands to both drives at the same time by giving
 * the group object instead of the one of the drive object. The next call
 * is two time more efficient than calling successively the fonction on
 * each drive, and this would also be true if the two drives were
 * connected to different COM ports (COM1/COM2) for example.
 */
err = dsa_power_on_s(grp, 10000);
if(err < 0)
    goto _error;

/* Start indexation on both axes. */
err = dsa_homing_start_s(grp, 10000);
if(err < 0)
    goto _error;

/*
 * We must now read the minimum and maximum values of each axis, but this
 * can obviously not been done with the group object, because every axis
 * of the group can have different values. In fact, none of the getter
 * functions (dsa_get_xxx) will accept a DSA_DRIVE_GROUP as its first
 * parameter.
 */
for(i = 0; i < 2; i++) {
    err = dsa_get_min_soft_position_limit_s(drv[i], &pos_min[i],
    DSA_GET_CURRENT, DSA_DEF_TIMEOUT);
    if(err < 0)
        goto _error;
    err = dsa_get_max_soft_position_limit_s(drv[i], &pos_max[i],
    DSA_GET_CURRENT, DSA_DEF_TIMEOUT);
    if(err < 0)
        goto _error;
}

/* Wait for the end of the movement of both axes. */
err = dsa_wait_movement_s(grp, 60000);
if(err < 0)
    goto _error;

/*
 * The main part of the program comes here, but we have created a
 * separated function called move_xy() to perform a movement and wait
 * until its end before returning. Here we will just generate the
 * random coordinates and pass them to the move_xy() function. After 100
 * iterations, we will jump to the (0,0) position and stop the program.
```

```
         */
        for(i = 0; i < 100; i++) {
            double x = pos_min[0] + (rand() * (pos_max[0] - pos_min[0])) / RAND_MAX;
            double y = pos_min[1] + (rand() * (pos_max[1] - pos_min[1])) / RAND_MAX;
            err = move_xy(grp, x, y);
            if(err < 0)
                goto _error;
        }
        err = move_xy(grp, 0.0, 0.0);
        if(err < 0)
            goto _error;

        /* Power off the motor. */
        err = dsa_power_off_s(grp, 60000);
        if(err < 0)
            goto _error;

        /* Close both connections. */
        for(i = 0; i < 2; i++) {
            err = dsa_close(drv[i]);
            if(err < 0)
                goto _error;
        }

        /* We should then destroy the group object. */
        err = dsa_destroy(&grp);
        if(err < 0)
            goto _error;

        /* We should then destroy the drive objects. */
        for(i = 0; i < 2; i++) {
            err = dsa_destroy(drv + i);
            if(err < 0)
                goto _error;
        }

        /* Success. */
        fprintf(stderr, "success.\n");
        return 0;

        /* Error handler. */
    _error:
        /*
         * First destroy the group if not already done, it is no more useful.
         */
        if(dsa_is_valid_drive_group(grp))
            dsa_destroy(&grp);

        /*
         * Then check the status of each drive object individually.
         */
        for(i = 0; i < 2; i++) {

            /* Is the drive pointer valid ? */
            if(dsa_is_valid_drive(drv[i])) {

                /* Is the drive open ? */
                bool open = 0;
                dsa_is_open(drv[i], &open);
                if(open) {
```

```
                /*
                 * We can check if a motor is switched on by reading the
                 * first bit (the lsb) of M60. To do that, we use the
                 * dsa_get_register_s() function. The variable that stores
                 * the value returned by dsa_get_register_s() has to be
                 * initialized correctly in order to properly handle the
                 * possibility of a failure in the request.
                 */
                int m60 = 1;
                dsa_get_register_s(
                    drv[i],             /* the drive object */
                    3,                  /* 3 = M register */
                    60,                 /* index */
                    0,                  /* subindex */
                    &m60,               /* returned value */
                    DSA_GET_CURRENT,    /* kind of value */
                    DSA_DEF_TIMEOUT     /* default timeout */
                );
                if(m60 & 0x00000001) {
                    /* Stop all movements. */
                    dsa_quick_stop_s(drv[i], 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(drv[i], 60000);
                    dsa_power_off_s(drv[i], 60000);
                }

                /* Close the connection. */
                dsa_close(drv[i]);
            }

            /*
             * Finally, release all resources to the OS.
             */
            dsa_destroy(drv + i);
        }
    }

    /* Print the first error that occured. */
    fprintf(stderr, "error %d: %s.\n", err, dsa_translate_error(err));
    return 1;
}

/*
 * move_xy() function.
 */
static int move_xy(DSA_DRIVE_GROUP *xy, double x, double y)
{
    /* The common variables must be defined. */
    int err;
    int i;

    /*
     * The first thing to do is to retrieve the individual DSA_DRIVE
     * contained in the xy group. We will need them because we want to assign
     * different positions to each of them. These drive objects could be
     * passed with additional parameters, but we want to keep the design
     * clean.
     */
    DSA_DRIVE *drv[2];
    for(i = 0; i < 2; i++) {
```

```
        err = dsa_get_group_item(xy, i, drv + i);
        if(err < 0)
            goto _error;
    }

    /*
     * We can now set the target position for both axes, but this time, we
     * will not write it into subindex zero, because we want the movement to
     * start synchronously on both axes. Instead, we will first write the
     * movement in the first buffer (subindex 1) and start both movements
     * with another function. This is the recommended method when doing
     * multi-axis movement.
     */
    for(i = 0; i < 2; i++) {
        err = dsa_set_target_position_s(drv[i], 1, (i ? y : x), DSA_DEF_TIMEOUT);
        if(err < 0)
            goto _error;
    }

    /*
     * The dsa_new_setpoint_s() function order the drive to use the
     * information contained in one of its buffer to start a new movement.
     * The second parameter of the method specifies which buffer to use (1-3
     * currently defined) while 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:
     *  - 0x0001: use the target position of the given buffer
     *  - 0x0002: use the profile velocity of the given buffer
     *  - 0x0004: use the profile acceleration of the given buffer
     *  - ...
     * In our case, we just use the buffer to store the new target position.
     * We will then use 0x0001 for the mask.
     * This function is applied to the whole group, which means that if the
     * two controllers are on the same bus, the two motors will start at the
     * same time, regardless of what happens on the PC.
     * With Turbo-ETEL-Bus (TEB), you can synchronize all drives together.
     * In this case, the movements really start exactly at the same time
     * (within one microsecond). However, in this example, we communicate
     * with the drives through the ETEL-Bus (ETB). With this bus, the two
     * motors will start within a delay of one millisecond.
     */
    err = dsa_new_setpoint_s(xy, 1, 0x0001, DSA_DEF_TIMEOUT);
    if(err < 0)
        goto _error;

    /* Wait for the end of the movement. */
    err = dsa_wait_movement_s(xy, 60000);
    if(err < 0)
        goto _error;

    /* Success. */
    return 0;

    /* Error handler. */
_error:
    /* Simply return the error code to the caller. */
    return err;
}
```

Third example:

```
/*
 * sample_3.c 1.01a
 *
 * Copyright (c) 2001 ETEL SA. All rights reserved.
 *
 * This software is confidential and proprietary information of ETEL SA
 * ("Confidential Information"). You shall not disclose such Confidential
 * Information and shall use it only in accordance with the terms of the
 * license agreement you entered into with ETEL.
 *
 * This software is provided "AS IS," without a warranty 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.
 *
 * This 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. Licensee represents and warrants that it will not use or
 * redistribute the software for such purposes.
 */

/*
 * 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 two ETEL position loop controller configured.
 *  - an ETEL-Bus-Lite or ETEL-Bus-Lite2 connection between the drive and
 *     the PC on COM1.
 *  - an ETEL-Bus connection between the two ETEL controllers.
 *  - the drives must have the axes number 0 (for the master) and 1 (slave).
 *  - k45 must be set to insure that position 0 is inside the valid range.
 *  - k34 and k35 must be set properly.
 */

/*** libraries ***/

/* Include the standard I/O library, utilities and character conversion. */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

/* Include the platform specific header for thread managment. */
#include <process.h>
/* Include the platform specific header for keyboard input. */
```

```
#include <conio.h>

/* Include EDI high level library version 2.0x. */
#include <dsa20.h>

/*** prototypes ***/

/*
 * This utility function is defined to perform a simple xy movement.
 */
static int move_xy(DSA_DRIVE_GROUP* xy, double x, double y);

/*
 * The following two functions define the threads which will run
 * independently. The exact parameter type of theses functions as well as the
 * return value are platform specific.
 */
static void display_thread(void *param);
static void keyboard_thread(void *param);

/*** variables ***/

/*
 * This global variable can be set by any thread to request the end of the
 * program. The main thread will listen to this variable at each cycle.
 */
static bool abort_req;

/*** functions ***/

/*
 * Entry point function.
 */
int main(char argc, int *argv[])
{
    /* Stores the min/max position of each axis. */
    double pos_min[2], pos_max[2];

    /* Definition of the drive and the group objects. */
    DSA_DRIVE *drv[2] = { NULL, NULL };
    DSA_DRIVE_GROUP *grp = NULL;

    /* Definition of some global variable. */
    int err;
    int i;

    /* Create two drive objects. */
    for(i = 0; i < 2; i++) {
        err = dsa_create_drive(drv + i);
        if(err < 0)
            goto _error;
    }

    /* Create the group object and set it up. */
    err = dsa_create_drive_group(&grp, 2);
    if(err < 0)
        goto _error;
    for(i = 0; i < 2; i++) {
        err = dsa_set_group_item(grp, i, drv[i]);
        if(err < 0)
            goto _error;
```

```
}

/* Create connections. */
err = dsa_open_u(drv[0], "etb:COM1:0");
if(err < 0)
    goto _error;
err = dsa_open_u(drv[1], "etb:COM1:1");
if(err < 0)
    goto _error;

/*
 * We will start the two threads before sending any command to the
 * controllers. The mechanism for starting the two thread is platform
 * dependant.
 */
if(_beginthread(display_thread, 0, grp) <= 0) {
    err = DSA_ESYSTEM;
    goto _error;
}
if(_beginthread(keyboard_thread, 0, grp) <= 0) {
    err = DSA_ESYSTEM;
    goto _error;
}

/* Power on both axes. */
err = dsa_power_on_s(grp, 10000);
if(err < 0)
    goto _error;

/* Start the indexation on both axes. */
err = dsa_homing_start_s(grp, 10000);
if(err < 0)
    goto _error;

/* Retrieve minimum / maximum positions. */
for(i = 0; i < 2; i++) {
    err = dsa_get_min_soft_position_limit_s(drv[i], &pos_min[i],
        DSA_GET_CURRENT, DSA_DEF_TIMEOUT);
    if(err < 0)
        goto _error;
    err = dsa_get_max_soft_position_limit_s(drv[i], &pos_max[i],
        DSA_GET_CURRENT, DSA_DEF_TIMEOUT);
    if(err < 0)
        goto _error;
}

/* Wait for the end of the movement of both axes. */
err = dsa_wait_movement_s(grp, 60000);
if(err < 0)
    goto _error;

/*
 * Main program loop - the program will run indefinitely until the ESC
 * or SPACE key will be hit by the user.
 */
while(!abort_req) {
    double x = pos_min[0] + (rand() * (pos_max[0] - pos_min[0])) / RAND_MAX;
    double y = pos_min[1] + (rand() * (pos_max[1] - pos_min[1])) / RAND_MAX;
    err = move_xy(grp, x, y);
    if(err < 0)
        goto _error;
```

```
    }
    err = move_xy(grp, 0.0, 0.0);
    if(err < 0)
        goto _error;

    /* Power off the motor. */
    err = dsa_power_off_s(grp, 60000);
    if(err < 0)
        goto _error;

    /* Close both connections. */
    for(i = 0; i < 2; i++) {
        err = dsa_close(drv[i]);
        if(err < 0)
            goto _error;
    }

    /* We should then destroy the group and the drive object. */
    err = dsa_destroy(&grp);
    if(err < 0)
        goto _error;
    for(i = 0; i < 2; i++) {
        err = dsa_destroy(drv + i);
        if(err < 0)
            goto _error;
    }

    /* Success. */
    fprintf(stderr, "success.\n");
    return 0;

    /* Error handler. */
_error:
    /* Destroy the group if not already done. */
    if(dsa_is_valid_drive_group(grp))
        dsa_destroy(&grp);

    /* Check the status of each drive object individually. */
    for(i = 0; i < 2; i++) {

        /* Is the drive pointer valid ? */
        if(dsa_is_valid_drive(drv[i])) {

            /* Is the drive open ? */
            bool open = 0;
            dsa_is_open(drv[i], &open);
            if(open) {

                /* Stop movements. */
                dsa_quick_stop_s(drv[i], 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(drv[i], 60000);
                dsa_power_off_s(drv[i], 60000);

                /* Close the connection. */
                dsa_close(drv[i]);
            }
            /* Finally, release all resources to the OS. */
            dsa_destroy(drv + i);
```

```
        }
    }

    /* Print the first error that occured. */
    fprintf(stderr, "error %d: %s.\n", err, dsa_translate_error(err));
    return 1;
}

/*
 * move_xy() function.
 */
static int move_xy(DSA_DRIVE_GROUP *xy, double x, double y)
{
    /* The common variables must be defined. */
    int err;
    int i;

    /* Get individual drive objects. */
    DSA_DRIVE *drv[2];
    for(i = 0; i < 2; i++) {
        err = dsa_get_group_item(xy, i, drv + i);
        if(err < 0)
            goto _error;
    }

    /* Set target positions. */
    for(i = 0; i < 2; i++) {
        err = dsa_set_target_position_s(drv[i], 1, (i ? y : x), DSA_DEF_TIMEOUT);
        if(err < 0)
            goto _error;
    }

    /* Start movement. */
    err = dsa_new_setpoint_s(xy, 1, 0x0001, DSA_DEF_TIMEOUT);
    if(err < 0)
        goto _error;

    /* Wait for the end of the movement. */
    err = dsa_wait_movement_s(xy, 60000);
    if(err < 0)
        goto _error;

    /* Success. */
    return 0;

    /* Error handler. */
_error:
    /* Simply return the error code to the caller. */
    return err;
}

/*
 * Display thread function.
 */
static void display_thread(void *param)
{
    /* the common variables must be defined */
    int counter = 0;
    int err;
    int i;
    /*
```

```
 * Casting of parameter to a group of pointer.
 */
DSA_DRIVE_GROUP *grp = (DSA_DRIVE_GROUP *)param;

/* Get individual drive objects. */
DSA_DRIVE *drv[2];
for(i = 0; i < 2; i++) {
    err = dsa_get_group_item(grp, i, drv + i);
    if(err < 0)
        goto _error;
}

/*
 * Start an infinite loop. This function will be stopped by the OS only
 * when the whole process will end.
 */
for(;;) {
    /*
     * Each 100 ms, we will display the status and the position of each
     * drive. As usually, the status structure must be initialized before
     * using it.
     */
    DSA_STATUS status[2] = { { sizeof(status) }, { sizeof(status) } };
    double pos[2];

    /* Get the position of each axis. */
    for(i = 0; i < 2; i++) {
        int err = dsa_get_position_actual_value_s(
            drv[i], pos + i, DSA_GET_CURRENT, 0);
        if(err)
            goto _error;
    }

    /*
     * Get the drive status. Theses functions don't generate any activity
     * on the communication channel. The status of each drive is always
     * kept up-to-date in the process memory, which means that theses
     * function are very effective and could be often called without
     * degrading performances.
     */
    for(i = 0; i < 2; i++) {
        int err = dsa_get_status(drv[i], status + i);
        if(err)
            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");
    }

    /*
```

```
             * We can now stop for 100ms; the following call is platform dependant
             */
            { extern __stdcall Sleep(int); Sleep(100); }
        }

        /* Error handler. */
_error:

        /* Ask the main thread to stop. */
        abort_req = 1;

        /* Print the first error that occured. */
           fprintf(stderr, "error in display (%d it) %d: %s.\n", counter, err,
dsa_translate_error(err));
}

/*
 * Keyboard thread function.
 */
static void keyboard_thread(void *param)
{
    /*
     * Casting of parameter to a group of pointer.
     */
    DSA_DRIVE_GROUP *grp = (DSA_DRIVE_GROUP *)param;

    /*
     * Infinite loop which waits for the keyboards inputs.
     */
    for(;;) {

        /*
         * Get a character in a system dependent way.
         */
        char c = tolower(getch());
        /*
         * Handle the key.
         */
        switch(c) {
        case '\33':
            /*
             * Pressing the ESC key will stop the process cleanly at the end
             * of the cycle. This is done by signaling the event to the main
             * process thread.
             */
            abort_req = 1;
            break;
        case ' ':
            /*
             * Pressing the SPACE key will generate an "emergency stop" and
             * stop the system immediatly, without waiting for the end of
             * the cycle. We don't check the error code because if the
             * function fails, there is no other action that could be taken.
             */
            abort_req = 1;
            dsa_quick_stop_s(grp, DSA_QS_PROGRAMMED_DEC,
                DSA_QS_BYPASS | DSA_QS_STOP_SEQUENCE, DSA_DEF_TIMEOUT);
            break;
        }
    }
}
```

4.3.2 Example in C of the use of a DSMAX

First example:

```
/*
 * dsmax_1.c 1.01a
 *
 * Copyright (c) 2001 ETEL SA. All rights reserved.
 *
 * This software is confidential and proprietary information of ETEL SA
 * ("Confidential Information"). You shall not disclose such Confidential
 * Information and shall use it only in accordance with the terms of the
 * license agreement you entered into with ETEL.
 *
 * This software is provided "AS IS," without a warranty 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.
 *
 * This 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. Licensee represents and warrants that it will not use or
 * redistribute the software for such purposes.
 *
 */

/*
 * This simple demo program shows how to make basic interpolated movements
 * on a X-Y axes system using ETEL's EDI libraries and the DSMAX multi-axis
 * board. The program will switch the motors on, send an indexation command,
 * make a simple movement, switch to the intepolation mode and interpolate a
 * G-code movement.
 */

/*
 * To run this demo without modification, you must have:
 *  - two linear motors with two ETEL's position loop controllers configured.
 *  - one DSMAX multi-axis board on the isa bus, configured like this:
 *    mem-address = 0xD0000, irq = 10
 *  - a Turbo-ETEL-Bus (TEB) connection between the drives and the DSMAX.
 *  - the drives must have the axis number 0 and 1.
 *  - k45 must be set to insure that position 0 is inside the valid range.
 *  - k34 and k35 must be set properly.
 */



/*** libraries ***/

/*
 * Include the standard I/O library.
 */
#include <stdio.h>
```

```
/*
 * Include EDI high level library version 2.0x.
 */
#include <dsa20.h>


/*** functions ***/

/*
 * Entry point function.
 */
int main(char argc, int *argv[])
{
    /* Define two drive objects. */
    DSA_DRIVE *x = NULL;
    DSA_DRIVE *y = NULL;

    /* Define a DSMAX object. This object represents a multi-axis board */
    DSA_DSMAX *dsmax = NULL;

    /*
     * Define an interpolation group. This group can be used like a normal
     * device or drive group. All drives within this group can also be
     * interpolated.
     */
    DSA_IPOL_GROUP *igrp = NULL;

    /* Definition of some global variable. */
    int err;

    /* Create the drive and DSMAX objects. */
    err = dsa_create_drive(&x);
    if(err < 0)
        goto _error;
    err = dsa_create_drive(&y);
    if(err < 0)
        goto _error;
    err = dsa_create_dsmax(&dsmax);
    if(err < 0)
        goto _error;

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

    /*
     * Associate the DSMAX to the interpolation group. To make interpolated
     * movement (like lines, circles and PVT), you need a multi-axis
     * board. When you ask for an interpolated movement on a group of axes,
     * the dsa20 library will use the DSMAX associated with the group.
     */
    err = dsa_set_dsmax(igrp, dsmax);
    if(err < 0)
```

```
    goto _error;

/*
 * Create connections. To have access to a DSMAX board on the isa bus,
 * you must specify the following string as url:
 *     etb:isa:<mem_address>,<irq_number>:<axis_number>
 * For the DSMAX, you must specify "*" as an axis number.
 * During the opening procedure, you can force the hardware reset of the
 * DSMAX and the drives. To do that, you must specify the following url:
 *     etb:isa:<mem_address>,<irq_number>,<reset_params>:<axis_number>
 * where <reset_params> can take the following values:
 *     r     to reset the DSMAX
 *     x     to reset all the drives
 *     rx    to reset the DSMAX and all the drives.
 */
err = dsa_open_u(x, "etb:isa:0xD0000,10,r:0");
if(err < 0)
    goto _error;
err = dsa_open_u(y, "etb:isa:0xD0000,10,r:1");
if(err < 0)
    goto _error;
err = dsa_open_u(dsmax, "etb:isa:0xD0000,10,r:*");
if(err < 0)
    goto _error;

/* Power on both axes. */
err = dsa_power_on_s(igrp, 10000);
if(err < 0)
    goto _error;

/* Start indexation on both axes. */
err = dsa_homing_start_s(igrp, 10000);
if(err < 0)
    goto _error;

/* Wait for the end of the movement of both axes. */
err = dsa_wait_movement_s(igrp, 60000);
if(err < 0)
    goto _error;

/* Make a movement to a specified point. */
err = dsa_set_target_position_s(x, 0, 0.01, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;
err = dsa_set_target_position_s(y, 0, 0.02, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Wait for the end of the movement on both axes. Here, we use the
 * interpolation group like a normal group. The DSMAX associated with
 * this group isn't taken into account.
 */
err = dsa_wait_movement_s(igrp, 60000);
if(err < 0)
    goto _error;

/*
 * Now we enter in interpolated mode. In this mode the position of the
 * motors is controlled by the multi-axis board. This is right only for
 * drives that are specified in the interpolation group (igrp). When you
```

```
 * are in this mode, you cannot use the dsa_set_target_position_x()
 * functions on the drives specified in the igrp. You must use the
 * dsa_ipol_xxx() functions instead to define the interpolated path. The
 * interpolator (the DSMAX) will drive the motors along this path.
 * Actually, the X-Y system is in the following absolute position (in
 * meters):
 *     x = 0.01, y = 0.02
 * When you enter into interpolated mode, the actual position becomes
 * the reference postition for the interpolation. For the interpolation
 * functions (dsa_ipol_xxx()), all coordinates are given in absolute
 * from this reference point. Practically, after the call to
 * dsa_ipol_begin_s()you are in the point:
 *     x = 0.0, y = 0.0
 * When you leave the interpolated mode, you come back to the previous
 * reference system.
 */
err = dsa_ipol_begin_s(igrp, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Define the maximum tengential speed and acceleration that the
 * interpolator must perform during the interpolation.
 * The tangential speed is set to 0.05 m/s.
 * The tangential acceleration and deceleration are set to 0.1 m/(s*s).
 */
err = dsa_ipol_tan_velocity_s(igrp, 0.05, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;
err = dsa_ipol_tan_acceleration_s(igrp, 0.1, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;
err = dsa_ipol_tan_deceleration_s(igrp, 0.1, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Add a linear segment to the interpolation path. After this function,
 * the interpolator must directly start the movement along this segment
 * and stop it at the end of the segment.
 */
err = dsa_ipol_line_2d_s(igrp, -0.005, 0.001, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Add another linear segment to the interpolation path.
 */
err = dsa_ipol_line_2d_s(igrp, 0.005, 0.001, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Start the concatenation of the interpolated segments. In this mode
 * all segments are processed at constant speed, without stoping between
 * segments. This means that when the interpolator processes the
 * previous segment, it does not decrease the speed at the end of the
 * segment. It jumps to the next segment without speed change.
 */
err = dsa_ipol_begin_concatenation_s(igrp, DSA_DEF_TIMEOUT);
if(err < 0)
```

```
    goto _error;

/*
 * Add a circular segment to the interpolation path. This segment is
 * defined by the destination point (second and third parameters) and
 * the center of the arc (fourth and fifth parameters) in the X-Y plane.
 * The circular segment is processed in the counter-clockwise direction.
 */
err = dsa_ipol_circle_ccw_c2d_s(igrp, 0.008, 0.002, 0.005, 0.006,
DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Add some segments to the interpolation path.
 */
err = dsa_ipol_line_2d_s(igrp, 0.012, 0.005, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;
err = dsa_ipol_circle_cw_c2d_s(igrp, 0.019, -0.002, 0.015, 0.001,
DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;
err = dsa_ipol_line_2d_s(igrp, 0.016, -0.006, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Stop the concatenated mode. After this command, the interpolator
 * stops the movement at the end of the previous segment and between each
 * following segments. Practically, the interpolator decreases the speed
 * in order to arrive at the end of the segment with a null speed. If you
 * don't call dsa_ipol_begin_concatenation_s(), the interpolator must
 * restart the movement at the beginning of the next segment, stop at its
 * end, restart at the beginning of the next one and so on for each
 * following segments.
 */
err = dsa_ipol_end_concatenation_s(igrp, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Add a new segment to the trajectory path.
 */
err = dsa_ipol_circle_ccw_c2d_s(igrp, 0.007, -0.003, 0.012, -0.003,
DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Go back to the concatenation mode. In this way, the interpolator
 * appends to the next segment without stoping the movement
 * between them.
 */
err = dsa_ipol_begin_concatenation_s(igrp, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Add a few segments to the trajectory path.
 */
```

```
err = dsa_ipol_circle_cw_c2d_s(igrp, 0.003, -0.003, 0.005, -0.003,
DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Stop concatenation.
 */
err = dsa_ipol_end_concatenation_s(igrp, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Just a few segments again.
 */
err = dsa_ipol_line_2d_s(igrp, -0.005, -0.003, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;
err = dsa_ipol_line_2d_s(igrp, -0.007, -0.005, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Wait for the end of the interpolated movement.
 */
err = dsa_ipol_wait_movement_s(igrp, 60000);
if(err < 0)
    goto _error;

/*
 * Leave the interpolation mode and come back to the previous reference
 * system.
 */
err = dsa_ipol_end_s(igrp, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Power off the motors.
 */
err = dsa_power_off_s(igrp, 60000);
if(err < 0)
    goto _error;

/* Close all connections. */
err = dsa_close(x);
if(err < 0)
    goto _error;
err = dsa_close(y);
if(err < 0)
    goto _error;
err = dsa_close(dsmax);
if(err < 0)
    goto _error;

/* We should then destroy the devices and group objects. */
err = dsa_destroy(&igrp);
if(err < 0)
    goto _error;
err = dsa_destroy(&x);
if(err < 0)
```

```
        goto _error;
    err = dsa_destroy(&y);
    if(err < 0)
        goto _error;
    err = dsa_destroy(&dsmax);
    if(err < 0)
        goto _error;

    /* Everything seems successful, we can exit now. */
    fprintf(stderr, "success.\n");
    return 0;

    /* Error handler. */
_error:
    /* Leave the interpolation mode. */
    if (dsa_is_ipol_in_progress(igrp))
        dsa_ipol_end_s(igrp, DSA_DEF_TIMEOUT);

    /* Destroy the group if not already done. */
    if(dsa_is_valid_drive_group(igrp))
        dsa_destroy(&igrp);

    /* Is the drive pointer valid ? */
    if(dsa_is_valid_drive(x)) {

        /* Is the drive open ? */
        bool open = 0;
        dsa_is_open(x, &open);
        if (open) {

            /* Stop all movements. */
            dsa_quick_stop_s(x, 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(x, 60000);
            dsa_power_off_s(x, 10000);

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

        /* And finally, release all resources to the OS. */
        dsa_destroy(&x);
    }

    /* Same operations for the second drive */
    if(dsa_is_valid_drive(y)) {

        /* Is the drive open ? */
        bool open = 0;
        dsa_is_open(y, &open);
        if (open) {

            /* Stop all movements. */
            dsa_quick_stop_s(y, 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(y, 60000);
            dsa_power_off_s(y, 10000);
```

```
            /* Close the connection */
            dsa_close(y);
        }

        /* And finally, release all resources to the OS. */
        dsa_destroy(&y);
    }

    /* Same operations for the DSMAX */
    if(dsa_is_valid_dsmax(dsmax)) {

        /* Is the drive open ? */
        bool open = 0;
        dsa_is_open(dsmax, &open);
        if (open) {

            /* Close the connection */
            dsa_close(dsmax);
        }

        /* And finally, release all resources to the OS. */
        dsa_destroy(&dsmax);
    }

    /* Print the first error that occured. */
    fprintf(stderr, "error %d: %s.\n", err, dsa_translate_error(err));
    return 1;
}
```

Second example:

```
/*
 * dsmax_2.c 1.01a
 *
 * Copyright (c) 2001 ETEL SA. All rights reserved.
 *
 * This software is confidential and proprietary information of ETEL SA
 * ("Confidential Information"). You shall not disclose such Confidential
 * Information and shall use it only in accordance with the terms of the
 * license agreement you entered into with ETEL.
 *
 * This software is provided "AS IS," without a warranty 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.
 *
 * This 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. Licensee represents and warrants that it will not use or
 * redistribute the software for such purposes.
 */
```

```
/*
 * This is a simple program that use the DSMAX multi-axis board to perform
 * an interpolated movememt 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 two ETEL's position loop controllers configured.
 *  - one DSMAX multi-axis board on the isa bus, configured like this:
 *    mem-address = 0xD0000, irq = 10
 *  - a Turbo-ETEL-Bus (TEB) connection between the drives and the DSMAX.
 *  - the drives must have the axes number 0 and 1.
 *  - k45 must be set to insure that position 0 is inside the valid range.
 *  - k34 and k35 must be set properly.
 */

/*** libraries ***/

/*
 * Include the standard I/O library.
 */
#include <stdio.h>

/*
 * Include EDI high level library version 2.0x.
 */
#include <dsa20.h>

/*
 * Include another EDI library to make some defines available.
 */
#include <dmd10.h>

/*** functions ***/

/*
 * Entry point function.
 */
int main(char argc, int *argv[])
{
    /* Define two drive objects. */
    DSA_DRIVE *x = NULL;
    DSA_DRIVE *y = NULL;

    /* Define a DSMAX object. */
    DSA_DSMAX *dsmax = NULL;

    /* Define a group that can be interpolated. */
    DSA_IPOL_GROUP *igrp = NULL;

    /*
     * Define vectors used to specify the destination point and velocity of a
     * PVT segment. Be careful to always set the size of the structures.
     */
    DSA_VECTOR destination = { sizeof(DSA_VECTOR) };
    DSA_VECTOR velocity = { sizeof(DSA_VECTOR) };

    /* Define some global variable. */
```

```
double time;
int err;

/* Create the drive and DSMAX objects. */
err = dsa_create_drive(&x);
if(err < 0)
    goto _error;
err = dsa_create_drive(&y);
if(err < 0)
    goto _error;
err = dsa_create_dsmax(&dsmax);
if(err < 0)
    goto _error;

/* Create the interpolation group object and set it up. */
err = dsa_create_ipol_group(&igrp, 2);
if(err < 0)
    goto _error;
err = dsa_set_group_item(igrp, 0, x);
if(err < 0)
    goto _error;
err = dsa_set_group_item(igrp, 1, y);
if(err < 0)
    goto _error;
err = dsa_set_dsmax(igrp, dsmax);
if(err < 0)
    goto _error;

/* Create connections. */
err = dsa_open_u(x, "etb:isa:0xD0000,10,r:0");
if(err < 0)
    goto _error;
err = dsa_open_u(y, "etb:isa:0xD0000,10,r:1");
if(err < 0)
    goto _error;
err = dsa_open_u(dsmax, "etb:isa:0xD0000,10,r:*");
if(err < 0)
    goto _error;

/* Power on both axes. */
err = dsa_power_on_s(igrp, 10000);
if(err < 0)
    goto _error;

/* Start indexation on both axes. */
err = dsa_homing_start_s(igrp, 10000);
if(err < 0)
    goto _error;

/* Wait for the end of the movement of both axes. */
err = dsa_wait_movement_s(igrp, 600000);
if(err < 0)
    goto _error;

/* Make a movement to a specified point. */
err = dsa_set_target_position_s(x, 0, 0.0, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;
err = dsa_set_target_position_s(y, 0, -0.03, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;
```

```
err = dsa_wait_movement_s(igrp, 60000);
if(err < 0)
    goto _error;

/*
 * Start of the trajectory
 */
printf("Start of the trajectory\n");

/*
 * Now we enter into interpolated mode.
 */
err = dsa_ipol_begin_s(igrp, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * PVT command is defined by its destination point, the velocity at this
 * point, and the time of the displacement.The trajectory defined here is
 * quite similar to the one described in the PVT chapter of the DSMAX
 * User's Manual.
 */

/*
 * Define the first PVT destination point, velocity and time:
 * x    = 10.0e-3 m
 * vx   = 50e-3 m/s
 * y    = 0 m
 * vy   = 0 m/s
 * time = 200.0e-3 s
 */
destination.val.dim.x = 10.0e-3;
destination.val.dim.y = 0;

velocity.val.dim.x = 50.0e-3;
velocity.val.dim.y = 0;

time = 200.0e-3;

/*
 * Add this PVT segment to the trajectory path. After this function call,
 * the interpolator will directly start the movement along this segment
 * and the next ones.
 */
err = dsa_ipol_pvt_s(igrp, &destination, &velocity, time, 60000);
if (err < 0)
    goto _error;

/*
 * Define the 2nd PVT destination point, velocity and time:
 * x    = 110.0e-3 m
 * vx   = 50e-3 m/s
 * y    = 0 m
 * vy   = 0 m/s
 * time = 2.0 s
 */
destination.val.dim.x = 110.0e-3;
destination.val.dim.y = 0;

velocity.val.dim.x = 50.0e-3;
velocity.val.dim.y = 0;
```

```
time = 2.0;

/*
 * Add this PVT segment to the trajectory path.
 */
err = dsa_ipol_pvt_s(igrp, &destination, &velocity, time, 60000);
if (err < 0)
    goto _error;

/*
 * Define and add some other PVT segments:
 */
destination.val.dim.x = 110.0e-3;
destination.val.dim.y = 0;
velocity.val.dim.x = 0;
velocity.val.dim.y = 50.0e-3;
time = 1.0;
err = dsa_ipol_pvt_s(igrp, &destination, &velocity, time, 60000);
if (err < 0)
    goto _error;

destination.val.dim.x = 110.0e-3;
destination.val.dim.y = 100.0e-3;
velocity.val.dim.x = 0;
velocity.val.dim.y = 50.0e-3;
time = 2.0;
err = dsa_ipol_pvt_s(igrp, &destination, &velocity, time, 60000);
if (err < 0)
    goto _error;

destination.val.dim.x = 110.0e-3;
destination.val.dim.y = 100.0e-3;
velocity.val.dim.x = -50.0e-3;
velocity.val.dim.y = 0;
time = 1.0;
err = dsa_ipol_pvt_s(igrp, &destination, &velocity, time, 60000);
if (err < 0)
    goto _error;

destination.val.dim.x = 10.0e-3;
destination.val.dim.y = 100.0e-3;
velocity.val.dim.x = -50.0e-3;
velocity.val.dim.y = 0;
time = 2.0;
err = dsa_ipol_pvt_s(igrp, &destination, &velocity, time, 60000);
if (err < 0)
    goto _error;

destination.val.dim.x = 10.0e-3;
destination.val.dim.y = 100.0e-3;
velocity.val.dim.x = 0;
velocity.val.dim.y = -50.0e-3;
time = 1.0;
err = dsa_ipol_pvt_s(igrp, &destination, &velocity, time, 60000);
if (err < 0)
    goto _error;
destination.val.dim.x = 10.0e-3;
destination.val.dim.y = 0;
velocity.val.dim.x = 0;
velocity.val.dim.y = -50.0e-3;
```

```
time = 2.0;
err = dsa_ipol_pvt_s(igrp, &destination, &velocity, time, 60000);
if (err < 0)
    goto _error;

destination.val.dim.x = 10.0e-3;
destination.val.dim.y = -10e-3;
velocity.val.dim.x = 0;
velocity.val.dim.y = 0;
time = 200.0e-3;
err = dsa_ipol_pvt_s(igrp, &destination, &velocity, time, 60000);
if (err < 0)
    goto _error;

/*
 * Wait for the end of the interpolated movement.
 */
err = dsa_ipol_wait_movement_s(igrp, 60000);
if(err < 0)
    goto _error;

/*
 * Leave the interpolation mode.
 */
err = dsa_ipol_end_s(igrp, DSA_DEF_TIMEOUT);
if(err < 0)
    goto _error;

/*
 * Power off the motors.
 */
err = dsa_power_off_s(igrp, 60000);
if(err < 0)
    goto _error;

/* Close all connections. */
err = dsa_close(x);
if(err < 0)
    goto _error;
err = dsa_close(y);
if(err < 0)
    goto _error;
err = dsa_close(dsmax);
if(err < 0)
    goto _error;

/* Destroy the objects. */
err = dsa_destroy(&igrp);
if(err < 0)
    goto _error;
err = dsa_destroy(&x);
if(err < 0)
    goto _error;
err = dsa_destroy(&y);
if(err < 0)
    goto _error;
err = dsa_destroy(&dsmax);
if(err < 0)
    goto _error;

/* Everything seems successful, we can exit now. */
```

```
    fprintf(stderr, "success.\n");
    return 0;

    /* Error handler. */
_error:
    /* Leave the interpolation mode. */
    if (dsa_is_ipol_in_progress(igrp))
        dsa_ipol_end_s(igrp, DSA_DEF_TIMEOUT);

    /* Destroy the group if not already done. */
    if(dsa_is_valid_drive_group(igrp))
        dsa_destroy(&igrp);

    /* Is the drive pointer valid ? */
    if(dsa_is_valid_drive(x)) {

        /* Is the drive open ? */
        bool open = 0;
        dsa_is_open(x, &open);
        if (open) {

            /* Stop all movements. */
            dsa_quick_stop_s(x, 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(x, 60000);
            dsa_power_off_s(x, 10000);

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

        /* And finally, release all resources to the OS. */
        dsa_destroy(&x);
    }

    /* Same operations for the second drive */
    if(dsa_is_valid_drive(y)) {

        /* Is the drive open ? */
        bool open = 0;
        dsa_is_open(y, &open);
        if (open) {

            /* Stop all movements. */
            dsa_quick_stop_s(y, 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(y, 60000);
            dsa_power_off_s(y, 10000);

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

        /* And finally, release all resources to the OS. */
        dsa_destroy(&y);
    }
```

```
    /* Same operations for the DSMAX */
    if(dsa_is_valid_dsmax(dsmax)) {

        /* Is the drive open ? */
        bool open = 0;
        dsa_is_open(dsmax, &open);
        if (open) {

            /* Close the connection */
            dsa_close(dsmax);
        }

        /* And finally, release all resources to the OS. */
        dsa_destroy(&dsmax);
    }

    /* Print the first error that occured. */
    fprintf(stderr, "error %d: %s.\n", err, dsa_translate_error(err));
    return 1;
}
```