



— The bright side of industry automation

OLTOM Card Driver Programming

© 2000–2010 OLTOM Engineering AB

Email: support@oltom.com

Version 1.58

Table of Contents

Introduction.....	4
Error Recovery and Timeout.....	4
Timeout and Retransmission.....	4
Addressing.....	4
Synchronization.....	4
Installation.....	5
Windows.....	5
Linux.....	5
Installed Files.....	5
Compiling and Linking.....	6
GNU C/MINGW.....	6
Microsoft Visual Studio C++.....	6
.NET and COM.....	8
Running or Debugging.....	9
API Calling Conventions.....	10
Introduction.....	10
Host Port Names.....	10
Data Types.....	10
Return Values and Error Codes.....	12
System Error Codes.....	12
OLTOM Device Error Codes.....	13
Synchronized Commands.....	14
Synchronization ID.....	14
Timing.....	14
Synchronization Disclaimer.....	14
Function Reference.....	15
oltom_openserial.....	15
oltom_open.....	15
oltom_close	15
oltom_getdriverversion	15
oltom_inquiry.....	15
oltom_setspeed.....	16
oltom_setacceleration.....	16
oltom_setspeedacceleration.....	16
oltom_resetcard.....	17
oltom_setcurrent.....	17
oltom_move.....	17
oltom_stop.....	18
oltom_go.....	18
oltom_synchstart.....	18
oltom_debuginfo.....	19
oltom_data.....	19
oltom_data4	19
oltom_readdata.....	20
oltom_arbitrarypacket	20
oltom_millisleep.....	20
oltom_settimeout.....	20
oltom_gettimeout	21
oltom_settretries.....	21
oltom_gettretries.....	21
oltom_setlogfile.....	21

oltom_setlogging.....	21
Examples.....	22
Programming in C.....	22
Programming in Visual Basic 6.....	24
DLL Redistribution License.....	26
Disclaimer.....	26

Introduction

The OLTOM products and software communicate with short packets, similar in concept to the Point to Point Protocol (PPP).

The packets can be encapsulated and sent on several different underlying transport technologies, such as asynchronous serial ports (RS232), Inter-Integrated Circuit (I2C), Universal Serial Bus (USB), Ethernet, and so on.

In version 1 of the packet format, the maximum total length of the packet, excluding framing and escaping, is 10 bytes.

For recovering from transmission errors a sequence number mechanism is used together with packet checksums. Lost or corrupted packets are automatically retransmitted.

To hide the complexity of the communication from the application programmer, all necessary functions are encapsulated in a stand-alone dynamic link library (DLL) with a well defined application programming interface API.

The API is the same on Windows and Linux platforms, and can be accessed from most programming environments that can call standard DLLs, such as C++, Visual Basic, .NET, Perl and so on.

For Windows development there is a .NET assembly, also containing COM mappings, for easy access from managed code environments, or COM clients.

Error Recovery and Timeout

For recovering from transmission errors a sequence number mechanism is used in combination with acknowledge packets. Lost or corrupted packets are automatically retransmitted if no acknowledge is received within the timeout period.

Timeout and Retransmission

Current default timeout before retransmission is 50 milliseconds. The default number of retransmissions before reporting an error is 5.

Both the timeout and the number of retransmissions can be changed via the API.

Addressing

One byte is used each for source and destination address. The special address 0xFF hexadecimal (255 decimal) is used as a broadcast address for special purposes. Thus, up to 255 (0x00-0xFE) different devices can be addressed within a single OLTOM system.

Special address limitation apply to I2C-systems. See the data sheet for the specific device for more information.

Note that several logical devices, each with its own address can be present on one physical card. With several devices on one card, all devices on one card have the same address prefix: The most significant bits of the address are the same for all devices on one card. The address byte only differs in the least significant bits.

Synchronization

The Move, Go, Stop and SetSpeed commands can be synchronized between motor controllers on the same card, and on separate cards. For synchronized start, stop or change speed the command to be synchronized is first sent to all controllers, then the synchronization command is broadcast to all controllers.

Installation

We recommend always downloading the latest version of the Card Driver from OLTOM.com.

Windows

For Windows, download the oltom-card-driver-X.X.msi (where X.X is the current version number) installation package, double click on the downloaded file and follow the instructions. The Win32 DLL and the .NET assembly will be installed and registered, both for .NET and COM access.

If you do not use .NET or COM, but only access the standard Win32 DLL, we recommend that you copy the oltom-card-driver.dll file to the same directory as your program. In that way you do not need to register the DLL with Windows. It is also very easy to upgrade the DLL: Just replace the file.

If you want to keep the oltom-card-driver.dll file a separate directory, add the DLL directory to the system path.

To set the path temporarily in a command window do:

```
set PATH=%PATH%; c:\Program Files\OLTOM\oltom-card-driver
```

To set the path permanently, right click My Computer or Computer, choose properties, select the advanced tab, click Environments Variables, edit Path, and add the directory to the end of the path.

Linux

For Linux, download the oltom-card-driver-X.X.zip (where X.X is the current version number) file. Unzip the installation bundle in the location of your choice, for example

```
/opt/OLTOM/oltom-card-driver
```

Either copy the DLL file liboltom-card-driver.so to the same directory as your program executable, or add the directory containing liboltom-card-driver.so to the linker search path.

Consult your distribution reference manual for instructions on how to add DLL search paths.

For example, to add the library search path on Redhat Enterprise Linux 5.x, CentOS 5.x and similar distributions, either create the file

```
/etc/ld.so.conf.d/oltom
```

containing the path to the OLTOM installation directory, for example:

```
/opt/OLTOM/oltom-card-driver
```

or add the path to the environment variable, either temporarily or permanently:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/OLTOM/oltom-card-driver
```

Installed Files

After installation you will have the following files in your installation directory. Please note that the exact files may vary between different versions of the DLL.

The header files necessary for compiling C, C++, and Visual Basic, as well as the DLL-library file itself will be placed into the directory:

```
liboltom-card-driver.so  
oltom-card-driver.bas  
oltom-card-driver.dll  
oltom-card-driver.h  
oltom-card-driver.lib  
oltom-card-driver.net.dll  
oltom-card-driver-programming-reference.pdf
```

Compiling and Linking

GNU C/MINGW

On Windows use the following command line (type all on one line):

```
"C:\Program Files\MinGW\bin\gcc" -I"C:\Program Files\OLTOM\oltom-card-driver" -L"C:\Program Files\OLTOM\oltom-card-driver" myprogram.c -loltom-card-driver
```

On Linux use the following command line (type all on one line):

```
gcc -I/opt/OLTOM/oltom-card-driver"-L"/opt/OLTOM/oltom-card-driver" myprogram.c -loltom-card-driver
```

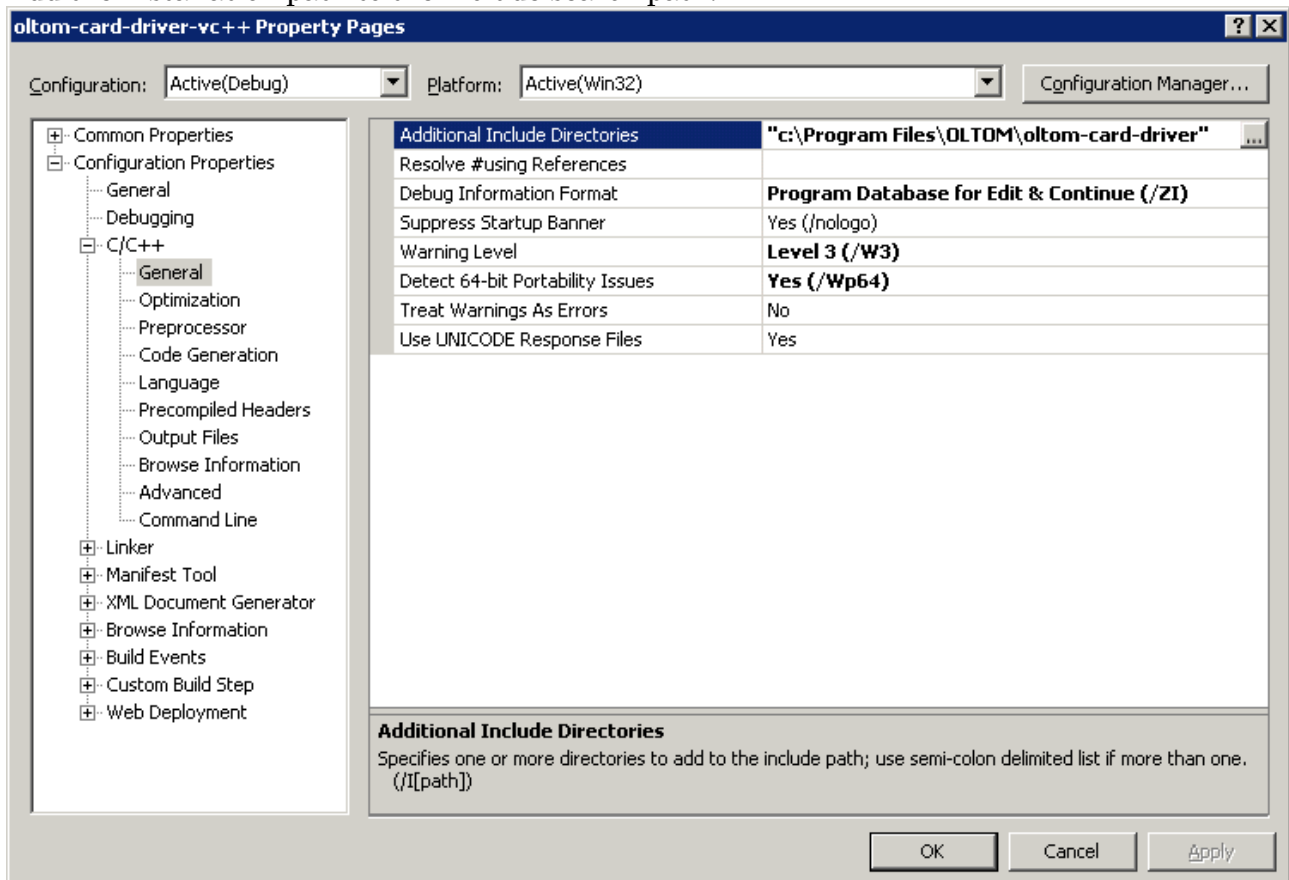
If using a development or make environment, set up include paths, library paths, and link dependencies as in the example above.

If the OLTOM card driver is installed in other than the suggested default directory, adjust the directory paths above accordingly.

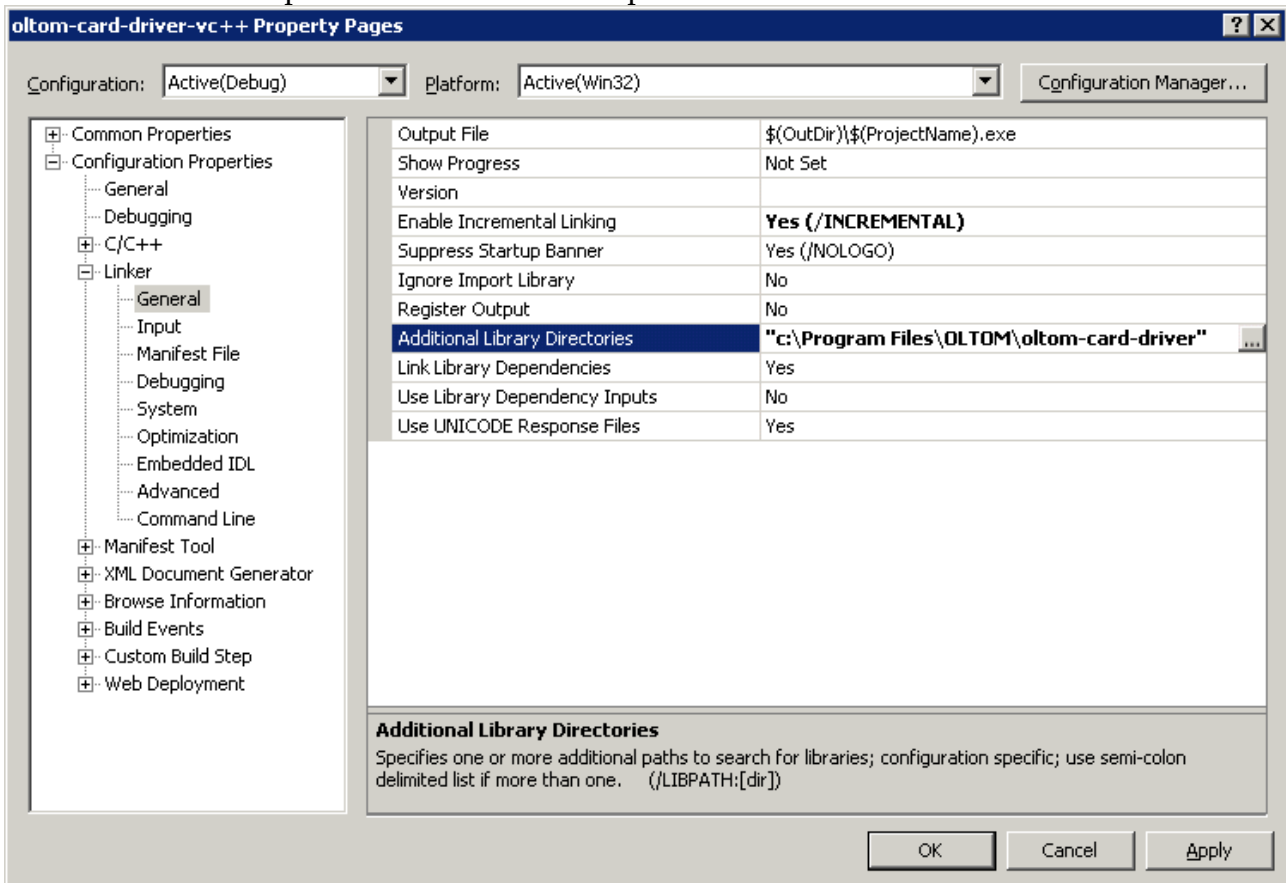
Microsoft Visual Studio C++

Note that the following example illustrates accessing the DLL directly with Win32 calling conventions. For accessing the DLL as a .NET managed code assembly, see the .NET section later.

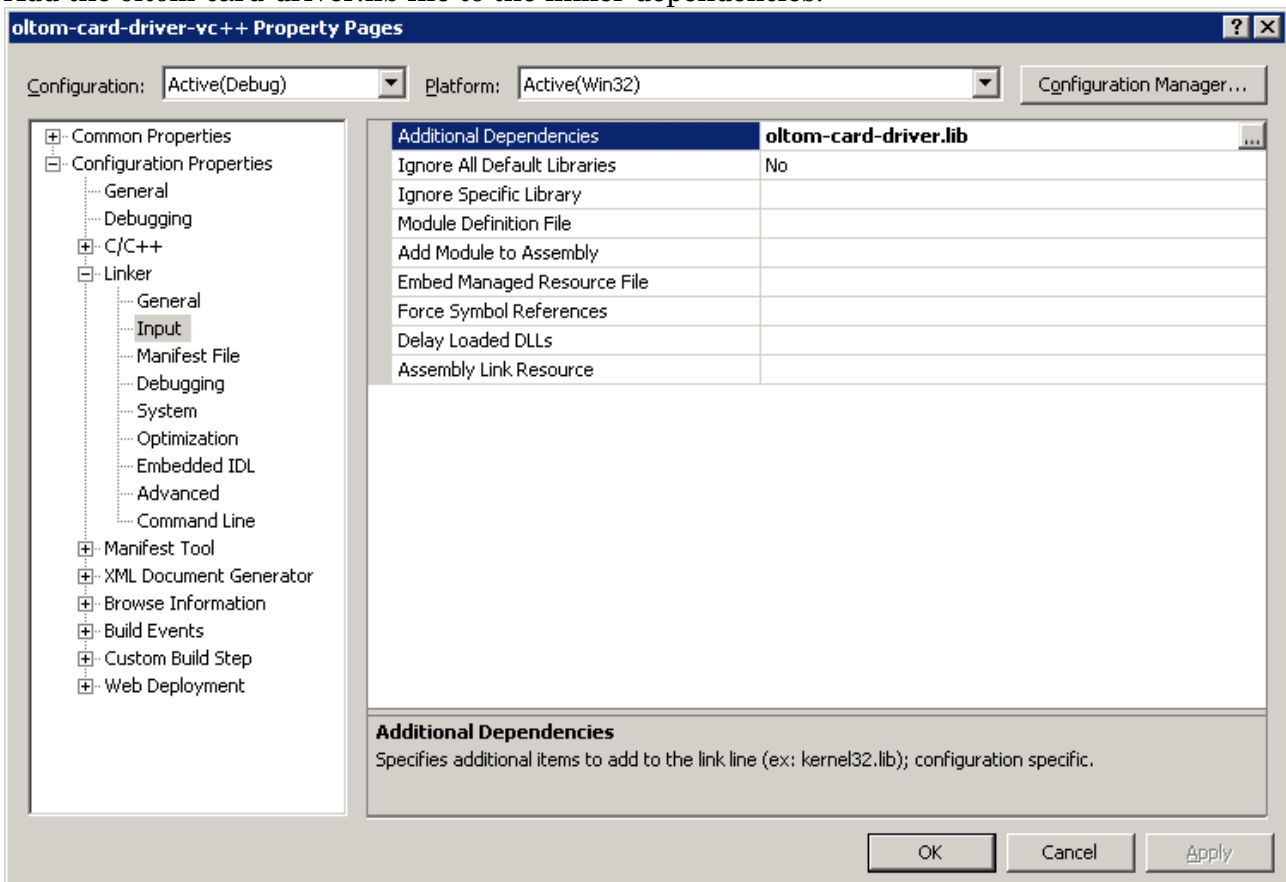
Add the installation path to the include search path:



Add the installation path to the linker search path:



Add the oltom-card-driver.lib file to the linker dependencies:



.NET and COM

For .NET and COM object access and programming a wrapper around the standard DLL is provided in the `oltom-card-driver.net.dll` assembly.

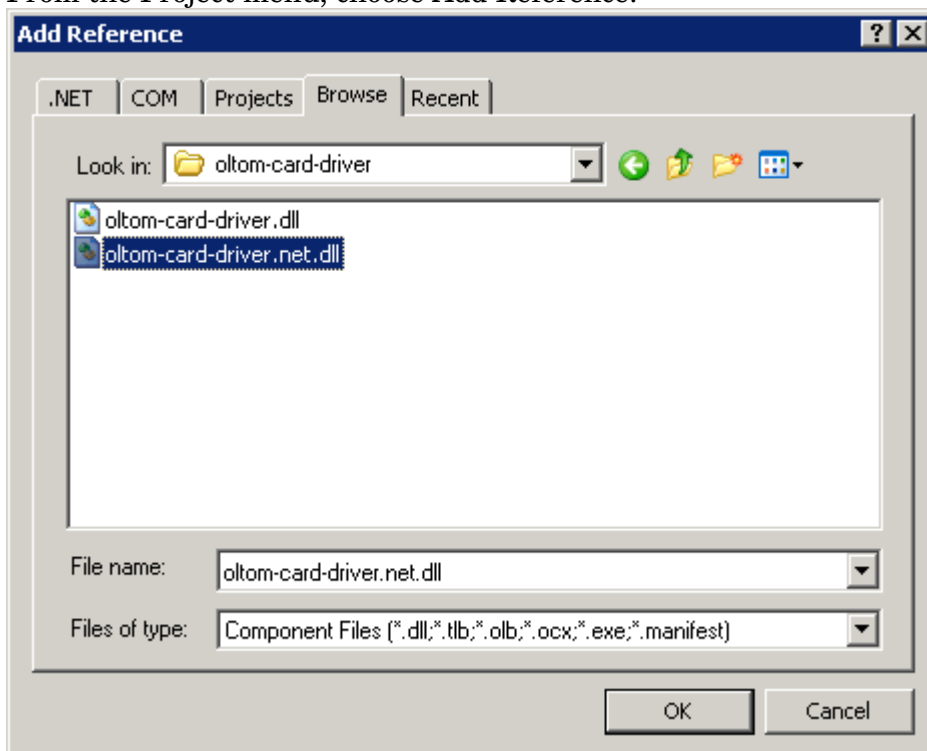
The wrapper exports the DLL functions as managed code objects. There are two different classes in the assembly, one with static function members and one with non-static class functions, for COM access. (COM calling conventions prevent the use of static member functions.)

Use the class with static members for ordinary .NET programming. All `oltom_` functions are directly accessible without instantiating the class.

The COM object is deprecated, and is only provided for backwards compatibility with older programs and code. Do not use the COM object for new projects that have can use .NET objects.

For development, just add a reference to the `oltom-card-driver.net.dll` assembly. The namespace used is `OLTOM`.

From the Project menu, choose Add Reference:



Then to open an instance using serial port 9 (COM9) in C++.NET do:

```
using namespace OLTOM;
...
System::Int32 h= CardDriver::oltom_open ("COM9");
```

In Visual Basic.NET do:

```
Dim h As Int32
h = OLTOM.CardDriver.oltom_open("COM9")
```

Note that both the `oltom-card-driver.dll` and `oltom-card-driver.net.dll` files must be accessible to the .NET application. Alternatively, the assembly can be put into the global assembly cache.

Running or Debugging

For running the program either copy the oltom-card-driver.dll file, (and the oltom-card-driver.net.dll file if using .NET) from the installation directory to the same directory as your program executable file (the .exe file); or add the OLTOM installation directory to your search path as described in the Installation section.

API Calling Conventions

Introduction

The dynamic link library (DLL) application programming interface is given in high level code with input and output parameters.

All calls use standard calling convention and should be compatible with all programming and scripting environments that are able to link and call external standard DLLs.

OLTOM provide header files mapping the DLL-entry points for C, C++, and Visual Basic. See the example section for usage.

Contact OLTOM Support if you would like header files with ready DLL-bindings for other languages. OLTOM may at its own discretion provide such a file.

Host Port Names

The name of the port the OLTOM card(s) is connected to must be given in the open command. Port names are system specific, see below.

Windows

On Windows COM1-COM9 work as device names as long as they are locally connected ports such as direct serial, and USB-RS232 ports.

Usually mapped, virtual serial ports such as RS232 over TCP/IP, remote desktop protocol (RDP) and so on, can be accessed by COM1-COM9 as well.

For special purposes it may be necessary to use a more specific device name.

For example, ports COM10-COM99 are not directly mapped in the Windows namespace, so the special Windows device name of `\\?/COMxx` has to be used. To open COM44, use `\\?/COM44`.

To explicitly use RDP-ports (terminal services), use portnames like `\\tsclient/comX`, e.g. `\\tsclient/com9` to open port COM9 on the remote client.

Linux/Unix

Use standard device names such as `/dev/ttyS0` for port names. Check your `/dev` directory and the system documentation for the correct device name.

Data Types

The following data types are used. The C equivalent parameter declaration is given.

String

All strings are passed and returned as *pointers* to null-terminated 8-bit ASCII byte vectors, exactly as standard C string conventions. If using .NET the data is automatically translate to and from the String type.

```
char * parameter;
```

Integer32

32-bit two-complement signed integer.

```
int parameter;  
assert (sizeof(int)==4);
```

Byte

8-bit unsigned integer from 0 to 255.

```
unsigned char parameter;
```

ByteVector

Pointer to byte sequence. Number of bytes is given in separate variable.

```
unsigned char parameter[];
```

OltomHandle

Integer32 handle for accessing a DLL instance.

```
oltom_handle parameter;
```

Return Values and Error Codes

The DLL functions usually return 0 (zero) on success. Exceptions exist, for example `oltom_open`, see the API reference for details.

On error, a *negative* value (minus `errno` in C environments) is returned for errors originating in the DLL or in the host operating system.

Errors originating in the OLTOM firmware on a card are returned as a positive error code. See the table OLTOM Device Error Codes below for more information. Note that not all errors can be returned by all OLTOM devices.

System Error Codes

Note that these are returned as *negative* values from the DLL functions. The system `errno` object is set to the *positive* error code. For example a host timeout (`ETIMEDOUT`) would be returned as `-110`.

```
#define EPERM          1 /* Operation not permitted */
#define ENOFILE       2 /* No such file or directory */
#define ESRCH        3 /* No such process */
#define EINTR        4 /* Interrupted function call */
#define EIO          5 /* Input/output error */
#define ENXIO        6 /* No such device or address */
#define E2BIG        7 /* Arg list too long */
#define ENOEXEC      8 /* Exec format error */
#define EBADF        9 /* Bad file descriptor */
#define ECHILD       10 /* No child processes */
#define EAGAIN       11 /* Resource temporarily unavailable */
#define ENOMEM       12 /* Not enough space */
#define EACCES       13 /* Permission denied */
#define EFAULT       14 /* Bad address */

#define EBUSY        16 /* strerror reports "Resource device" */
#define EEXIST       17 /* File exists */
#define EXDEV        18 /* Improper link (cross-device link?) */
#define ENODEV       19 /* No such device */
#define ENOTDIR      20 /* Not a directory */
#define EISDIR       21 /* Is a directory */
#define EINVAL       22 /* Invalid argument */
#define ENFILE       23 /* Too many open files in system */
#define EMFILE       24 /* Too many open files */
#define ENOTTY       25 /* Inappropriate I/O control operation */

#define EFBIG        27 /* File too large */
#define ENOSPC       28 /* No space left on device */
#define ESPIPE       29 /* Invalid seek (seek on a pipe?) */
#define EROFS        30 /* Read-only file system */
#define EMLINK       31 /* Too many links */
#define EPIPE        32 /* Broken pipe */
#define EDOM         33 /* Domain error (math functions) */
#define ERANGE       34 /* Result too large (possibly too small) */

#define EDEADLOCK    36 /* Resource deadlock avoided */

#define ENAMETOOLONG 38 /* Filename too long */
#define ENOLCK       39 /* No locks available */
#define ENOSYS       40 /* Function not implemented */
#define ENOTEMPTY    41 /* Directory not empty */
#define EILSEQ       42 /* Illegal byte sequence */

#define ETIMEDOUT    110 /* Operation timed out */
```

OLTOM Device Error Codes

The following errors are returned as positive integers. The appropriate constants below are defined in the header files. The ASCII character corresponding to each error code is also listed.

For example, a busy device would return the constant `oltom_busy` (integer 66, which is the ASCII code for 'B').

Note that `oltom_busy` is the only code that should be returned during normal operations.

P oltom_invalid_packet

Invalid Packet Type. The packet contains an invalid/unimplemented type byte.

C oltom_invalid_command

Invalid Command. The command packet contains an invalid/unimplemented command type byte.

A oltom_invalid_argument

Invalid Argument. The arguments given after the command type are invalid, or out of range.

D oltom_invalid_destination

Invalid Destination address. The command is addressed to a device that is not present on this card.

B oltom_busy

Busy. The device is busy executing a previous command and cannot receive another one until it the previous command has finished executing.

L oltom_locked

Locked. The device is locked or in an invalid position and cannot execute the command.

V oltom_invalid_protocol

Invalid Protocol Version. The device does not recognize the protocol version used.

E oltom_internal_error

Internal Error. The device encountered an internal program error while executing a command. Contact your support representative.

Synchronized Commands

The Move, Go, Stop and SetSpeed commands can be synchronized between motor controllers on the same card, and on separate cards attached to the same bus. For synchronized start, stop or change speed the command to be synchronized is first sent to all controllers. Then the synchronization command is broadcast to all controllers.

For example, to start continuous movement on a motor controller at address 0x5d on one card, and a controller at address 0x42 on another card:

1. Call `oltom_go` with address 0x5d, ID=1 (setup movement, but do not execute)
2. Call `oltom_go` with address 0x42, ID=1 (setup movement, but do not execute)
3. Call `oltom_synchstart` with ID=1. Both motors will start within 1 ms of each other.

Synchronization ID

In version 1.x of the API only one synchronization ID is supported (ID number 1).

Timing

As the `synchstart` command is broadcast on the I2C bus, the maximum time difference between device starts is 1 millisecond.

Synchronization Disclaimer

Note that there is no acknowledge of devices connected over I2C after the `synchstart` command. Communications errors that cause lost packets will prevent devices on different cards to start at the same time. The PC will receive an acknowledge for the `synchstart` command from the first card only.

Do not rely on the synchronization feature for critical, error sensitive applications needing fail-safe functionality.

It is the responsibility of the system designer to implement fail-safe features in hardware outside of the OLTOM software.

Also, check the return values from all commands that should be synchronized. An error indicates that the command has not been set up properly, and that the device will not start when the `synchstart` command is set. The application software should stop and inform the user of the error.

Function Reference

oltom_openserial

String **DeviceName**.
Integer32 **Bitrate**

Returns a positive or zero oltom_handle on success.
Returns -errno (always negative) on host system error.

See section Host Port Names for more information on the device name.

oltom_open

String **DeviceName**

Returns a positive or zero oltom_handle on success.
Returns -errno (always negative) on host system error.

Maps to oltom_open_serial and 115200 bits per second for backward compatibility.

oltom_close

OltomHandle **InstanceHandle**

Returns zero on success.
Returns -errno (always negative) on host system error.

Close a handle opened by oltom_open. The port will be closed, read and write threads will be terminated before returning.

oltom_getdriverversion

Returns a null-terminated ASCII string with the version number of the loaded DLL.

oltom_getversion

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**
Byte **TypeOfInfo**

Returns zero on success.
Returns -errno (always negative) on host system error.
Returns enum oltom_nak (always positive) on OLTOM device error or busy status

Ask device to send version information. The information is returned in a separate data packet, read with oltom_readdata. The return packet contains up to four bytes, according to TypeOfInfo:

- 0 Return product ID. Byte
- 1 Return hardware revision. Three bytes (a.b.c)
- 2 Return firmware revision. Three bytes (a.b.c)
- 3 Return hardware serial number. Four bytes unsigned integer, most significant first.

oltom_inquiry

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**

Returns zero on success. (The device is idle and ready to accept commands.)
Returns -errno (always negative) on host system error.
Returns enum oltom_nak (always positive) on OLTOM device error or busy status.

Inquire status for device. This function sends an inquire command to check the device status.

oltom_setspeed

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**
Integer32 **SpeedInStepsPerSecond**
Integer32 **DelayedSynchronizedExecutionID**

Returns zero on success.
Returns -errno (always negative) on host system error.
Returns enum oltom_nak (always positive) on OLTOM device error or busy status.

Set speed in steps per second for motor devices.

Set DelayedSynchronizedExecutionID to 0 for immediate execution. (No synchronized speed change is possible in version 1.x of the API.)

oltom_setacceleration

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**
Integer32 **AccelerationInStepsPerSecondSquared**

Returns zero on success.
Returns -errno (always negative) on host system error.
Returns enum oltom_nak (always positive) on OLTOM device error or busy status.

Set acceleration in steps per second squared for motor devices. The retardation will be set to the same value as well.

Use acceleration 0 for constant speed.

oltom_setspeedacceleration

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**
Integer32 **SpeedInStepsPerSecond**
Integer32 **AccelerationInStepsPerSecondSquared**

Returns zero on success.

Returns -errno (always negative) on host system error.

Returns enum oltom_nak (always positive) on OLTOM device error or busy status.

Two functions for the price of one: Does both setspeed and setacceleration in one function call. Less packets are sent to the device, typically this call will save several tens of milliseconds compared to first calling setspeed and then setacceleration.

Use acceleration 0 for constant speed.

Set DelayedSynchronizedExecutionID to 0 for immediate execution. (No synchronized speed change is possible in version 1.x of the API.)

oltom_resetcard

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**

Returns zero on success.

Returns -errno (always negative) on host system error.

Returns enum oltom_nak (always positive) on OLTOM device error or busy status.

Reset card to power on state

Note: This will reset all devices on a card with several devices. I.e. on a PWM3 card both motor drivers will be stopped and reset.

oltom_setcurrent

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**
Integer32 **CurrentInMilliAmpere**

Returns zero on success.

Returns -errno (always negative) on host system error.

Returns enum oltom_nak (always positive) on OLTOM device error or busy status.

Set the winding current. Current is sent as milliamperes. 1200 is 1.2 A. See the device documentation for more information on maximum and minimum values.

oltom_move

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**
Integer32 **StepsToMove**
Integer32 **DelayedSynchronizedExecutionID**

Returns zero on success.

Returns -errno (always negative) on host system error.

Returns enum oltom_nak (always positive) on OLTOM device error or busy status.

Move given number of steps, using the maximum speed specified with setspeed, the acceleration, and the retardation specified with setacceleration.

Positive steps are in the forward direction, negative steps are in the backward direction. Check the device documentation for exact behavior.

Maximum number of steps in protocol version 1 can be from -8388607 to 8388606 (24 bit signed integer).

Set DelayedSynchronizedExecutionID to 0 for immediate execution and to the synchronization ID for delayed synchronized execution.

oltom_stop

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**
Integer32 **DelayedSynchronizedExecutionID**

Returns zero on success.

Returns -errno (always negative) on host system error.

Returns enum oltom_nak (always positive) on OLTOM device error or busy status.

Stop current movement. The device will stop immediately without retardation.

Note: Mechanical inertia may cause the device to continue moving after a stop command. Do not use this function for e.g. emergency braking or for implementing failsafe functions.

Set DelayedSynchronizedExecutionID to 0 for immediate execution and to the synchronization ID for delayed synchronized execution.

oltom_go

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**
Integer32 **Direction**
Integer32 **DelayedSynchronizedExecutionID**

Returns zero on success.

Returns -errno (always negative) on host system error.

Returns enum oltom_nak (always positive) on OLTOM device error or busy status.

Go unlimited number of steps. The device will accelerate to the currently given speed and continue running. Use oltom_stop to end movement.

Set Direction=1 for movement in the forward direction, and set Direction=-1 (negative) for movement in the backward direction. Check the device documentation for exact behavior.

Set DelayedSynchronizedExecutionID to 0 for immediate execution and to the synchronization ID for delayed synchronized execution.

oltom_synchstart

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress** (see note below)
Integer32 **SourceAddress**
Integer32 **DelayedSynchronizedExecutionID**

Returns zero on success.

Returns -errno (always negative) on host system error.

Returns enum oltom_nak (always positive) on OLTOM device error or busy status.

Start previously sent commands with the same DelayedSynchronizedExecutionID on all devices connected to the same bus.

In version 1.x of the API, only one DelayedSynchronizedExecutionID (number 1) is supported.

Note that the DestinationAddress is the address of the card that broadcast the synchronization command. If only devices on the same card should be synchronized, use the address of that card.

oltom_debuginfo

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**
Byte **TypeOfInfo**

Returns zero on success.

Returns -errno (always negative) on host system error.

Returns enum oltom_nak (always positive) on OLTOM device error or busy status

Ask device to send debug information

This function is present to aid customer troubleshooting. It is only to be used in cooperation with an OLTOM support engineer. Do not make any assumptions about the result of this call.

oltom_data

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**
Integer32 **NumberOfBytesToSend**
ByteVector **DataToBeSent**

Returns zero on success.

Returns -errno (always negative) on host system error.

Returns enum oltom_nak (always positive) on OLTOM device error or busy status

Send raw data bytes in a packet. Usually for integration with third party devices requiring command strings.

oltom_data4

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**
Integer32 **NumberOfBytesToSend**
Byte **Data1**
Byte **Data2**
Byte **Data3**
Byte **Data4**

Returns zero on success.

Returns -errno (always negative) on host system error.

Returns enum oltom_nak (always positive) on OLTOM device error or busy status

Send up to four raw data bytes in a packet. Usually for integration with third party devices requiring command strings.

oltom_readdata

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**
Integer32 **MaximumDataLength**
ByteVector **ReceivedData**

Returns zero on success.

Returns -errno (always negative) on host system error.

Read a raw data packet sent by other device. Wait for an incoming data packet and read up to MaximumDataLength bytes into ReceivedData.

oltom_arbitrarypacket

OltomHandle **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**
Byte **PacketType**
Byte **PacketLength**,
ByteVector **PacketPayload**

Returns zero on success.

Returns -errno (always negative) on host system error.

Returns enum oltom_nak (always positive) on OLTOM device error or busy status

Send an arbitrary packet type. Only for use in special customized solutions. Do not use to access basic features such as move, read data etc.

oltom_millisleep

Integer32 **MillisecondsToSleep**

Sleep for the specified milliseconds.

Note that the actual time slept can be off by several tens of milliseconds on Windows.

oltom_settimeout

OltomHandle **InstanceHandle**
Integer32 **MillisecondsBeforeRetransmission**

Returns zero on success.

Returns -errno (always negative) on host system error.

Set timeout in milliseconds for packet retransmission. This is used to tune the performance of the OLTOM error recovery algorithm.

The default is suitable for host computers with locally attached OLTOM devices via local USB, I2C or serial ports.

For high latency environments, i.e. terminal services, serial over TCP/IP etc, the timeout may have to be increased to avoid retransmissions and/or communications failure.

For a dedicated control computer with real time scheduling and local devices, the timeout can be decreased, for lower latency in error reporting.

oltom_gettimeout

OltomHandle **InstanceHandle**

Returns current timeout in milliseconds on success.

Returns -errno (always negative) on host system error.

oltom_settretries

OltomHandle **InstanceHandle**
Integer32 **NumberOfRetransmissions**

Returns zero on success.
Returns -errno (always negative) on host system error.

Set number of retransmissions before reporting error.

The default number of retransmissions is suitable for local high quality connections. An error will typically be returned to the calling program within 250 ms of a major failure.

For very noisy, bad quality connections, i.e. running over an overloaded Internet connection, the number of retries can be increased.

oltom_gettretries

OltomHandle **InstanceHandle**

Returns current number of retransmissions before failing and reporting error.
Returns -errno (always negative) on host system error.

oltom_setlogfile

String **FileName**

Returns zero on success.
Returns -errno (always negative) on host system error.

Redirect log to file. If no file is explicitly set, the log messages will be written to stderr. (If the program is run in a command line window, the output will show up in the window.)

oltom_setlogging

Integer32 **LogLevel**

Returns zero on success.
Returns -errno (always negative) on host system error.

Set log verbosity: 0 no logging, 1-7 increasing verbosity.

Examples

Programming in C

The following program opens communications with an OLTOM PWM3 card connected to serial port COM2 on a Windows host.

The card address is set to 5C hex (using the card DIP switch).

The program sets the speed for the *second* motor on the card (address 5E) to 500 steps per second, and the acceleration to 1000 steps per second squared. It then moves 2500 steps forward, and 2500 steps backwards.

During movement the program busy waits on the motor, polling the status every 100 milliseconds.

For the sake of clarity only the first return error check has been implemented.

```

#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <oltom-card-driver.h>

int main () {
    const char * const testport= "COM2"; // Second serial port
    oltom_byte dest= 0x5e; // Second motor on PWM3 card @ address 0x5c
    oltom_byte src= 0x5c; // Address of serial port on card
    int retval= -1; // return code

    printf ("Trying to open port: %s\n", testport);
    oltom_handle h= oltom_open (testport);

    if (h < 0) {
        perror (testport);
        return errno;
    }
    // retval should be 0 on success
    retval= oltom_inquiry (h, dest, src);
    // Error check
    if (retval < 0) {
        // Host or DLL error
        perror ("Host error: ");
        exit (errno);
    } else if (retval == oltom_busy) {
        printf ("Card busy executing command.\n");
        exit (-1);
    } else if (retval > 0) {
        printf ("Card command error: %c\n", retval);
        exit (-1);
    }
    // Set speed 500 steps/s, Accelerate at 1000 steps/s^2
    retval= oltom_setspeed (h, dest, src, 500, 0);
    retval= oltom_setacceleration (h, dest, src, 1000);
    // Move 2500 steps forward with above speed & acceleration
    retval= oltom_move (h, dest, src, 2500, 0);
    // Busy wait. Poll every 0.1 second until movement done.
    while (oltom_busy == oltom_inquiry (h, dest, src)) {
        oltom_millisleep (100);
    }
    // Return to original position. (Move 2500 steps backward.)
    retval= oltom_move (h, dest, src, -2500, 0);
    // Busy wait. Poll every 0.1 second until movement done.
    while (oltom_busy == oltom_inquiry (h, dest, src)) {
        oltom_millisleep (100);
    }
    return 0;
}

```

Programming in Visual Basic 6

Important note: The examples below are based on Visual Basic 6, where the int data type is 16 bits wide. The Visual Basic 6 long type is 32 bits wide, and corresponds to the driver Integer32 data type. Note that later version of Visual Basic change the size of the int type.

All public functions in the driver are declared in a module, the oltom-card-driver.bas file, included with the installation package.

Below is an extract from the .bas file for reference:

```
Public Declare Function _
oltom_openserial _
Lib "oltom-card-driver.dll" _
(ByVal devicename As String, ByVal Btrate as Long) _
As Long

Public Declare Function _
oltom_inquiry _
Lib "oltom-card-driver.dll" _
(ByVal handle As Long, ByVal destadr As Byte, ByVal originadr As Byte) _
As Long
```

To use the functions, include the oltom-card-driver.bas module file in your project, and call the functions.

For calling the functions, declare a handle and the addresses. In the example below the addresses are declared as constants and the handle as a variable:

```
Public oltom_handle as long
Const dest = &H5E ' Second motor on PWM3 card @ address 0x5c
Const src = &H5C ' Address of serial port on card
```

The following program works in the same way as the previous C program example. It opens communications with an OLTOM PWM3 card connected to serial port COM2 on a Windows host.

The card address is set to 5C hex (using the card DIP switch).

The program sets the speed for the *second* motor on the card (address 5E) to 500 steps per second, and the acceleration to 1000 steps per second squared. It then moves 2500 steps forward, and 2500 steps backwards.

During movement the program busy waits on the motor, polling the status every 100 milliseconds.

For the sake of clarity full return error check has not been implemented, only basic error checking is done.

```

Public Function MoveMotors (dest as Byte, src as Byte) as long
Dim testport as string
Dim retval as integer

testport = "COM2"

oltom_handle = oltom_open(testport)

if oltom_handle < 0 Then
  MsgBox "COM Error: " & oltom_handle
  MoveMotors = oltom_handle ' Return error code
  Exit Function
End If

' retval should be 0 on success, if not motor busy->exit function
retval= oltom_inquiry (h, dest, src)

if retval <> 0 Then
  MoveMotors = retval ' Return the value of Busy motor
  Exit Function
End If

' Set speed 500 steps/s, Accelerate at 1000 steps/s^2
retval= oltom_setspeed (oltom_handle, dest, src, 500, 0)
retval= oltom_setacceleration (oltom_handle, dest, src, 1000)
' Move 2500 steps forward with above speed & acceleration
retval= oltom_move (h, dest, src, 2500, 0)

' Busy wait
Do
  DoEvents ' So program execution does not lock during while loop
While oltom_inquiry (h, dest, src) <> 0

' Return to original position. (Move 2500 steps backward.)
retval= oltom_move (h, dest, src, -2500, 0)

' Busy wait
Do
  DoEvents ' So program execution does not lock during while loop
While oltom_inquiry (h, dest, src) <> 0

MoveMotors = 0 ' Return 0, succesfull movements

End Function

```

DLL Redistribution License

As an OLTOM customer you are granted a non-exclusive, non-transferable, limited license to redistribute the OLTOM DLL component, specifically the oltom-card-driver.dll file and the oltom-card-driver.net.dll file, with your software or hardware for interfacing with OLTOM systems, or for interfacing with other systems using OLTOM intellectual property.

No other redistribution, publication, sublicensing or other use of OLTOM software is permitted under this license.

Additional licensing may be available, contact OLTOM if you have any special requirements.

Disclaimer

We believe that the information contained herein was accurate in all respects at the time of printing. OLTOM Engineering AB cannot, however, assume any responsibility for errors or omissions in this text. Also note that the information in this document is subject to change without notice and should not be construed as a commitment by OLTOM Engineering AB or its partners.