



— The bright side of industry automation

OLTOM LabVIEW Interface Manual

© 2000–2010 OLTOM Engineering AB

Email: support@oltom.com

Version 2.1

Table of Contents

Introduction.....	3
Timeout and Retransmission.....	3
Addressing.....	3
Synchronization.....	3
Installation.....	4
Windows.....	4
Installed Files.....	4
Compiling and Linking.....	5
VI.s.....	5
VI Calling Conventions.....	6
Introduction.....	6
Host Port Names.....	6
Return Values and Error Codes.....	7
System Error Codes.....	7
OLTOM Device Error Codes.....	8
Synchronized Commands.....	9
Synchronization ID.....	9
Synchronization Disclaimer.....	9
VI Reference.....	10
oltom_openserial.vi.....	10
oltom_open.vi.....	10
oltom_close.vi.....	10
oltom_getdriverversion.vi.....	10
oltom_getversion.vi.....	10
oltom_inquiry.vi.....	11
oltom_setspeed.vi.....	11
oltom_setacceleration.vi.....	11
oltom_setspeedacceleration.vi.....	12
oltom_resetcard.vi.....	12
oltom_setcurrent.vi.....	12
oltom_move.vi.....	13
oltom_stop.vi.....	13
oltom_go.vi.....	13
oltom_synchstart.vi.....	14
oltom_debuginfo.vi.....	14
oltom_data.vi.....	14
oltom_data4.vi.....	15
oltom_readdata.vi.....	15
oltom_arbitrarypacket.vi.....	15
oltom_millisleep.vi.....	16
oltom_settimeout.vi.....	16
oltom_gettimeout.vi.....	16
oltom_settretries.vi.....	16
oltom_gettretries.vi.....	16
oltom_setlogfile.vi.....	17
oltom_setlogging.vi.....	17
Examples.....	18
VI Programming.....	18
DLL Redistribution License.....	19
Disclaimer.....	19

Introduction

The main design goal for the drivers delivered by OLTOM is ease of use. The PWM3 stepper motor card supports a number of different operating systems and programming languages.

A problem with all communication, for example RS-232, is the risk of corrupted messages. A garbled command message could be fatal to a system.

The PWM3 communication protocol is sophisticated with several security features. Emphasis is placed on high reliability. All packets have data security features, and the sender receives an explicit acknowledge for all messages. This handling also enables the ability to reliably address cascaded PWM3 cards through the I2C bus.

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, LabVIEW, .NET, Perl and so on. For a more comprehensive understanding of the underlying DLL please read the documentation.

The LabVIEW development environment from National Instruments is a graphical, user friendly, programming environment for development of automation and measurement systems on ordinary PCs. LabVIEW programming uses a concept called Virtual Instruments (VI). The OLTOM LabVIEW Driver is delivered with a number of VIs that are easy to use and understand.

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

OLTOM Card Driver and VIs

To be able to use the OLTOM VIs the OLTOM DLL driver needs to be installed first. We recommend downloading the latest version of the Card Driver from OLTOM.com. For a comprehensive explanation of the driver see the DLL documentation.

For Windows, download the oltom-card-driver.msi 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.

After installation you will have the following files in your installation directory (usually C:\Program Files\OLTOM\oltom-card-driver). Please note that the exact files may vary between different versions of the DLL.

```
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
```

The VIs for interfacing to the DLL are installed in the subdirectory VI (usually C:\Program Files\OLTOM\oltom-card-driver\VI). One VI is provided for each DLL call.

DLL Path for VIs

LabVIEW must be able to find the oltom-card-driver.dll: Either copy the oltom-card-driver.dll file from the installation directory to the same directory as your program, or 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.

Redistribution of LabVIEW applications

If a LabVIEW application is compiled as an EXE-file, the oltom-card-driver.dll must be distributed with the application. The suggested installation directory for the DLL is in the same directory as the application EXE-file.

VI Calling Conventions

Introduction

The VI programming interface is adapted from the DLL with input and output parameters suitable for LabVIEW. All VI's use standard calling convention.

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.

LabVIEW for 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`.

Return Values and Error Codes

The VI's 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 VI. The system `errno` object is set to the *positive* error code. For example a host timeout would be returned as -110.

```
1      /* Operation not permitted */
2      /* No such file or directory */
3      /* No such process */
4      /* Interrupted function call */
5      /* Input/output error */
6      /* No such device or address */
7      /* Arg list too long */
8      /* Exec format error */
9      /* Bad file descriptor */
10     /* No child processes */
11     /* Resource temporarily unavailable */
12     /* Not enough space */
13     /* Permission denied */
14     /* Bad address */

16     /* strerror reports "Resource device" */
17     /* File exists */
18     /* Improper link (cross-device link?) */
19     /* No such device */
20     /* Not a directory */
21     /* Is a directory */
22     /* Invalid argument */
23     /* Too many open files in system */
24     /* Too many open files */
25     /* Inappropriate I/O control operation */

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

36     /* Resource deadlock avoided */

38     /* Filename too long */
39     /* No locks available */
40     /* Function not implemented */
41     /* Directory not empty */
42     /* Illegal byte sequence */

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 integer 66, which is the ASCII code for 'B'.

Note that integer 66 is the only code that should be returned during normal operations.

P = 80

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

C =67

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

A =65

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

D =68

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

B =66

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

L =76

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

V =86

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

E =69

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 VI 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. Use VI oltom_go with address 0x5d, ID=1 (setup movement, but do not execute)
2. Use VI oltom_go with address 0x42, ID=1 (setup movement, but do not execute)
3. Use VI oltom_synchstart with ID=1. Both motors will start synchronized.

Synchronization ID

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

Synchronization Disclaimer

Note that there is no acknowledge of devices connected over I2C after the synchstart command. Communications error that cause lost packets will prevent devices on different cards to start at the same time. From the first card connected to the PC the synchstart command will always cause an acknowledge if received and processed correctly.

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 VI oltom_syncstart command is set. The application software should stop and inform the user of the error.

VI Reference

oltom_openserial.vi

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.vi

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.vi

Integer32 **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.vi

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

oltom_getversion.vi

Integer32 **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.vi

Integer32 **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.vi

Integer32 **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.vi

Integer32 **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.vi

Integer32 **InstanceHandle**
Integer32 **DestinationAddress**
Integer32 **SourceAddress**
Integer32 **SpeedInStepsPerSecond**
Integer32 **AccelerationInStepsPerSecondSquared**
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.

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.vi

Integer32 **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.vi

Integer32 **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.vi

Integer32 **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.vi

Integer32 **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.vi

Integer32 **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.vi

Integer32 **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.vi

Integer32 **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.vi

Integer32 **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 .vi

Integer32 **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.vi

Integer32 **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 .vi

Integer32 **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.vi

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.vi

Integer32 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.vi

Integer32 InstanceHandle

Returns current timeout in milliseconds on success.

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

oltom_setretries.vi

Integer32 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_getretries.vi

Integer32 InstanceHandle

Returns current number of retransmissions before failing and reporting error.

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

oltom_setlogfile.vi

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.vi

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

VI Programming

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

Using the simple front panel a port name can be given, a speed from 1 to 3000 steps per second can be set, and movement commands can be executed.

Simple LED lights indicate the status of the commands (success or error).

Finally a close button releases the connection and closes the serial port.



Illustration 1: Simple LabVIEW front panel for controlling motor. The motor is at address 0x5D and the PC at 0x5C, using serial port 9 (com9).

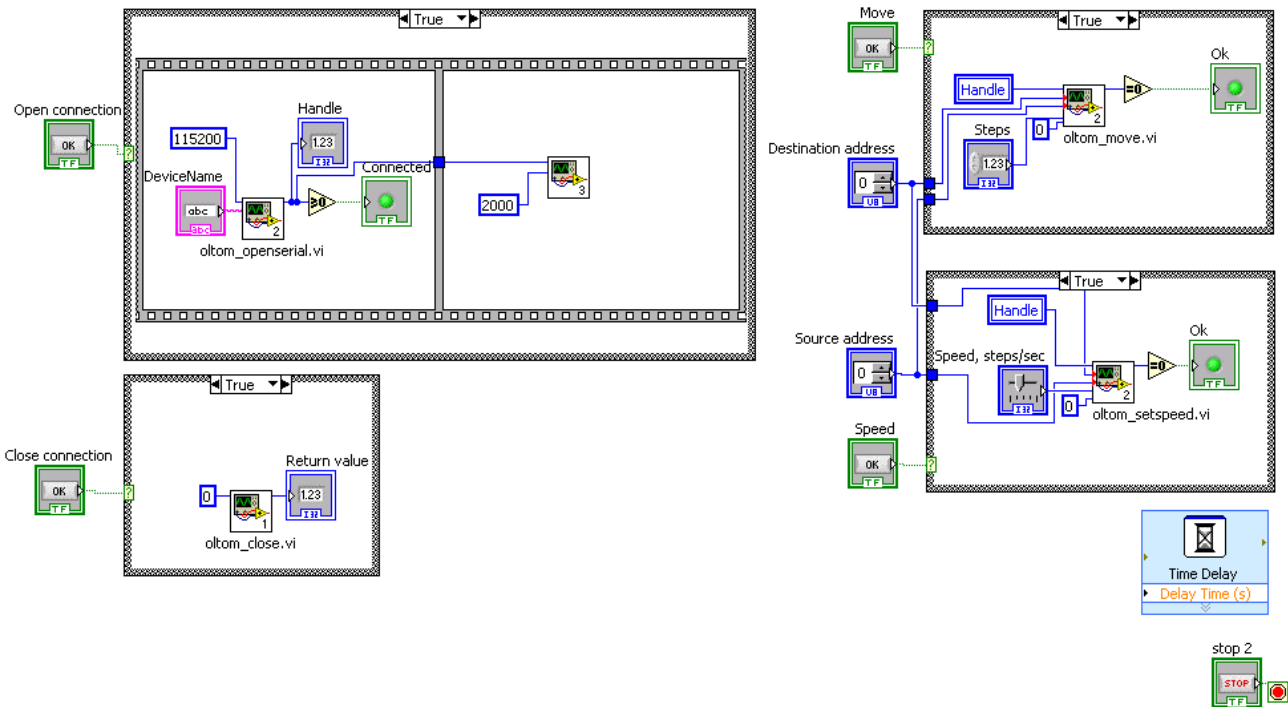


Illustration 2: The four sub-VIs for the example front panel. Clockwise from the upper left corner: Open port and get handle from DLL. Move the specified number of steps. Set speed. Close connection to DLL and release serial port.

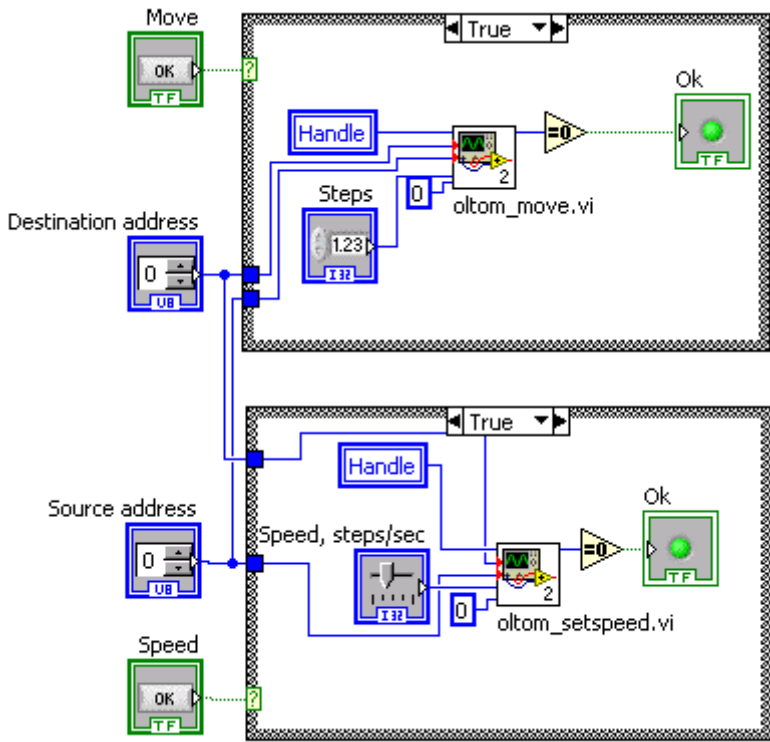


Illustration 3: The top VI sends a move command with the given number of steps. The bottom VI sets the speed.

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.