EPOS

Positioning Controller

Documentation

Linux shared library Integration into Eclipse C++/QT

maxon document rel1836

1 Table of contents

2 Table of figures

3 Introduction

This documentation “Linux shared library integration into C++ Eclipse/QT” explains how the ‘Linux 32-Bit shared library’ should be integrated into your application. This document should simplify the programming of the control software based on Linux/Ubuntu distribution.
This document should be treated as the quick start guide and does not contain informations about the full set of the library functions. A complete overview of all existing functions is available in the document “Epos command library”.
This documentation is intended to cover most applications in automatisation. It is based on the experience of maxon motor control. maxon motor control certifies that the content of this library is correct according his best knowledge.

This documentation refers to the devices EPOS and EPOS2. In order to simplify the document, only the notation EPOS is used.

The latest edition of these documentation “Linux shared library integration into QT/Eclipse”, additional documentation and software to the EPOS positioning controller are free of charge available under http://www.maxonmotor.com category <Service & Downloads> or in the maxon motor e-shop http://shop.maxonmotor.com.

4 How to use this guide

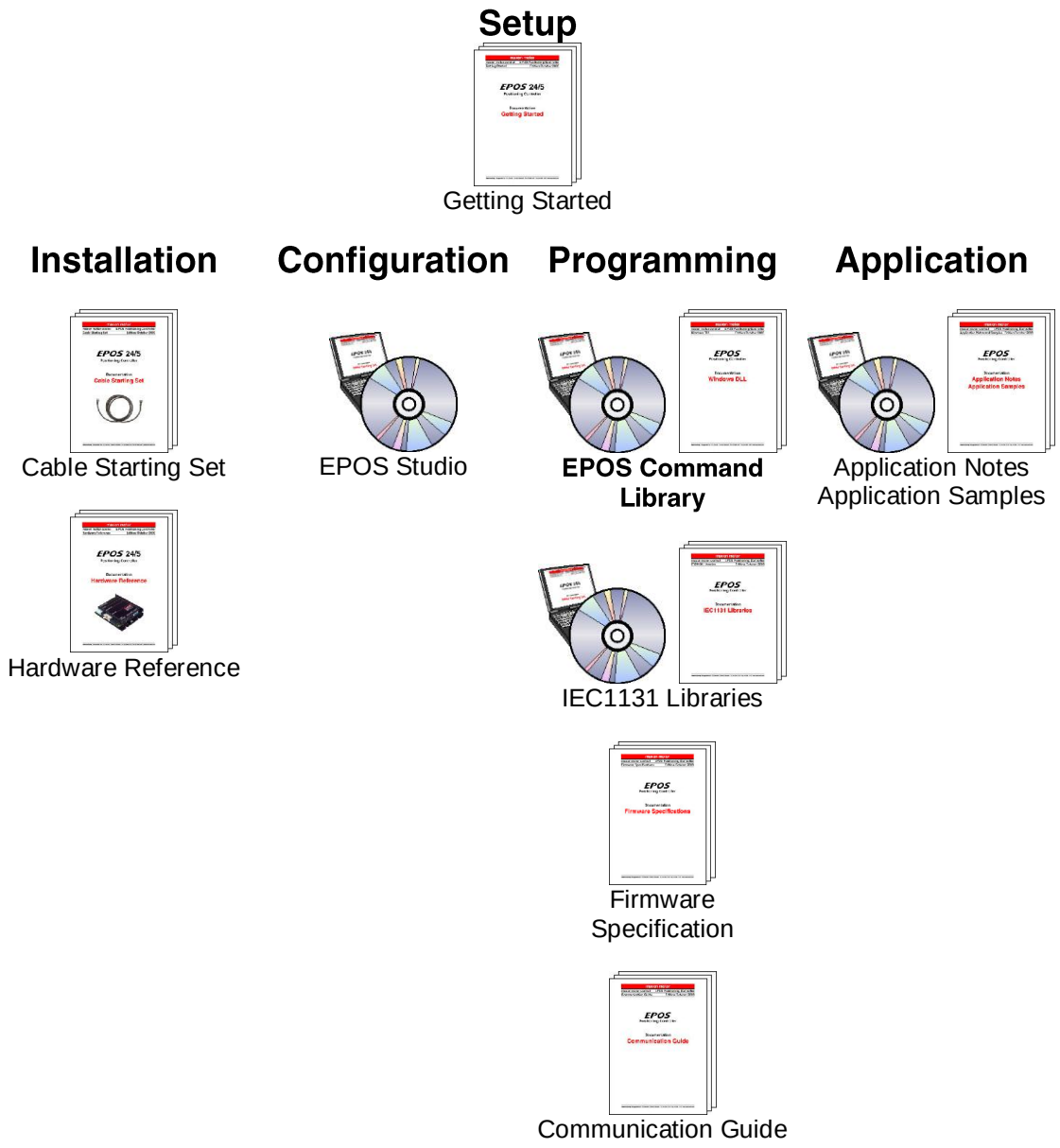

Figure 1: EPOS documentation hierarchy

5 Including Library Functions

5.1 General Information

The library 'libEposCmd.so' is an implementation of the RS232 and USB protocol for the communication between EPOS and a personal computer. Using this library is a simple way to develop your own application.

This library should properly run on each Linux 32-bit operating system with the kernel 2.6 and supporting ftd2xx library running on the x86 platform.

Tested with the following distributions:

- Ubuntu 10.04,10.10, 11.04, 11.10, 12.04, 12.10

You can include it into different C++ programming environments.

Tested with the following environments:

- Eclipse IDE for C/C++ Developers Helios with QT C++ Eclipse integration plugin

All the EPOS commands are implemented and they can be called directly from your own program. You do not have to care about the protocol details. The only thing you have to ensure is a correct communication setting of the port.

The library is generating data frames, sending and receiving these frames is the task of this communication library.

The library 'libEposCmd.so' offers now slightly limited set of EPOS commands (in comparison to the windows version), but the main functionality is supported. Missing is the parameter export/import functionality and GUI related functions (e.g. VCS_OpenDeviceDlg).

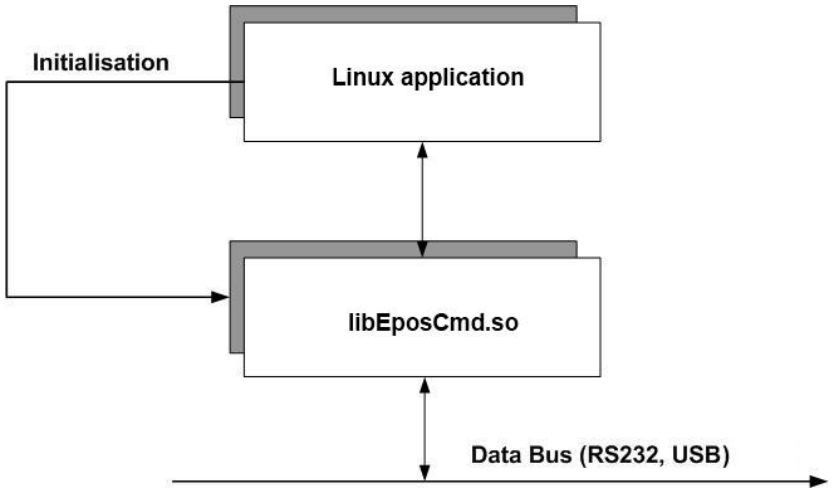

Figure 2: Library hierarchy

For your own program, you have to include this library.

5.2 Development tools

- Eclipse IDE for C/C++ Developer, min. Helios, Service Release 1
  - http://www.eclipse.org
- QT 4.7 SDK For Linux/X11 32-bit
  - http://qt.digia.com
- QT C++ Eclipse Integration (optional)
  - http://qt.digia.com

5.3 Required Libraries

- EPOS Command library
  - pthread
  - stdc++
  - libc
  - dl
  - ftd2xx
    - FTDI driver for USB communication with EPOS2 http://www.ftdichip.com

- QT Demo EPOS using Linux shared library
  - QT Framework 4.7
  - pthread
  - X11

5.4 Including

The following chapter describes how to include all library functions in your own Linux program. The way in which you do that, depends on the compiler and on the programming environment you use. Thus, we are going to explain the procedure to include the library based on some examples of the most popular programming tool.

In order to have a correctly working communication, you have to include the library

libEposCmd.so.<major>.<minor>.<rev>.0

to your programming environment. Sample file name: libEposCmd.4.7.3.0. You have to copy this file to the working directory of your system.

To configure the library 'libEposCmd.so.<major>.<minor>.<rev>.0' you have to use the function "VCS_OpenDevice". The available arguments you can find in the "EPOS Command library" document. For a correct communication, you have to select the baud rate and the timeout. Use the function "VCS_SetProtocolStackSettings". You have to do this before you can execute any EPOS command!
At the end of your program, you have to close all opened ports.

For more detailed information about the initialisation procedure, have a look at the following chapter 'Programming'.

5.5 FTD library installation

FTD library is used for the USB communication. This release was tested with version 1.1.12 of the library.

The latest version can be downloaded directly from manufacturer website. It is delivered with some Linux distributions too.

http://www.ftdichip.com/Drivers/D2XX.htm

1. Installation instructions

   a) Download and extract the *tar.gz file into a temporal directory (for example~/temp)
   b) Copy the shared library libftd2xx.so.1.1.12 file into the /usr/local/lib directory
      - sudo cp libftd2xx.so.1.1.12 /usr/local/lib (this action requires the root privileges)
      - It is necessary to create two soft links to this library in order to work properly. From the /usr/local/lib folder and as root, execute the following commands
        - cd /usr/local/lib
        - sudo ln -s libftd2xx.so.0.1.12 libftd2xx.so.0
        - sudo ln -s libftd2xx.so.0 libftd2xx.so
      - It is also necessary to create a soft link to this library into the /usr/lib folder. Again, as root, from the /usr/lib folder, execute the following command
        - cd /usr/lib
        - sudo ln -s /usr/local/lib/libftd2xx.so.1.1.12 libftd2xx.so
   c) It is necessary to include a new rule in the udev server. To do that, as root, copy the EPOS "99-ftdi.rules" file in the /etc/udev/rules folder
   d) Re-start the udev service to take into account the changes
      - sudo /etc/init.d/udev restart

5.6 Eclipse C++ with CDT plugin Project

You need the following files to include the library to the programming environment of Eclipse C++':

Definitions.h: Constant definitions and declarations of the library functions
libEposCmd.so.<major>.<minor>.<rev>.0: EPOS Linux Shared library

The following steps are necessary to include the library correctly:

First Step: You have to copy libeEposCmd.so.<major>.<minor>.<rev>.0 file in the directory listed in LD_LIBARARY_PATH environment variable. In the development phase you can use the current working directory, but then you have to use the delivered shell script (Demo_Linux.sh) to start the application. Please note, that the shared library location have to be listed in the LD_LIBRARY_PATH environment variable even if the executable and the shared library are located in the same directory!

Example:

Modify the LD_LIBRARY_PATH environment variable and add the /opt/lib directory.

```
LD_LIBRARY_PATH=/opt/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

please make sure, that this variable will be persistent in your system!

More about you will find in the Linux documentation (LD_LIBRARY_PATH) and man pages (ldconfig(8)).

Copy the libEposCmd.so.<major>.<minor>.<rls> to the /opt/lib

```
sudo cp ./libEposCmd.so.<major>.<minor>.<rls>  /opt/lib
```

To simplify the versioning of the library it will be necessary to create softlink's (symbolic links).

- libEposCmd.so.<major>.<minor>.<rev>.0 (original file)
- libEposCmd.so.0 (softlink)
- libEposCmd.so (softlink)

```
cd /opt/lib
sudo ln -s libEposCmd.so.<major>.<minor>.<rev>.0 libEposCmd.0
sudo ln -s libEposCmd.0 libEposCmd.so
```

optionally: sudo ldconfig –n /opt/lib

Second Step: The header file 'Definitions.h' has to be included into the program code. Use the instruction ***#include "Definitions.h"***.

Third Step: The file 'libEposCmd.so' has to be added to the project. For that purpose you have to open the menu point 'Properties' of the menu 'Project'. Select 'Libraries' in the tab control on the right side. Add the file 'libEposCmd.so' using 'Add external library' button. The figure shows the sample program version of this dialog.

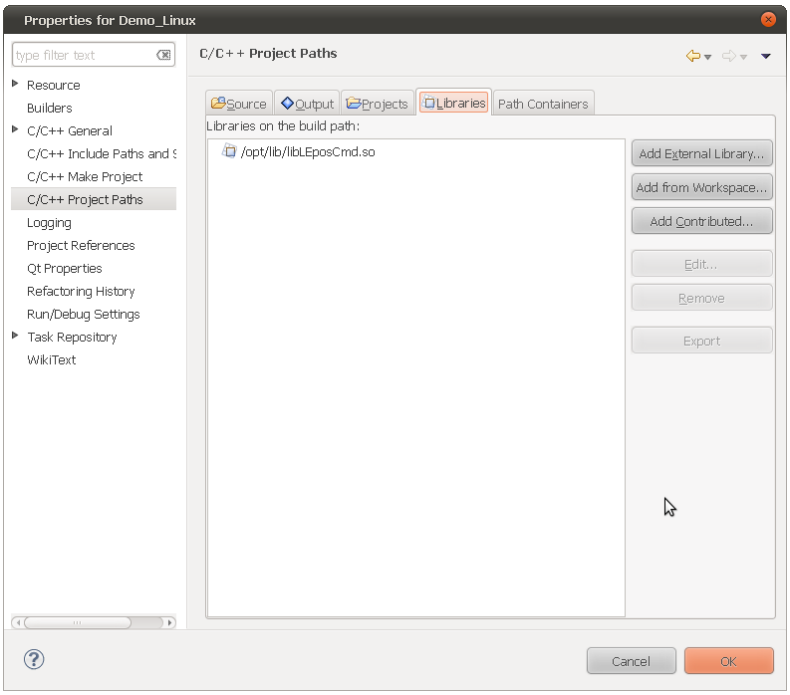

Figure 3: Project settings of Eclipse C++

In the case you are using the QT project. You will have to add the "LIBS" section to your .pro file with the path to the library (for example /opt/lib) and library name -lEposCmd

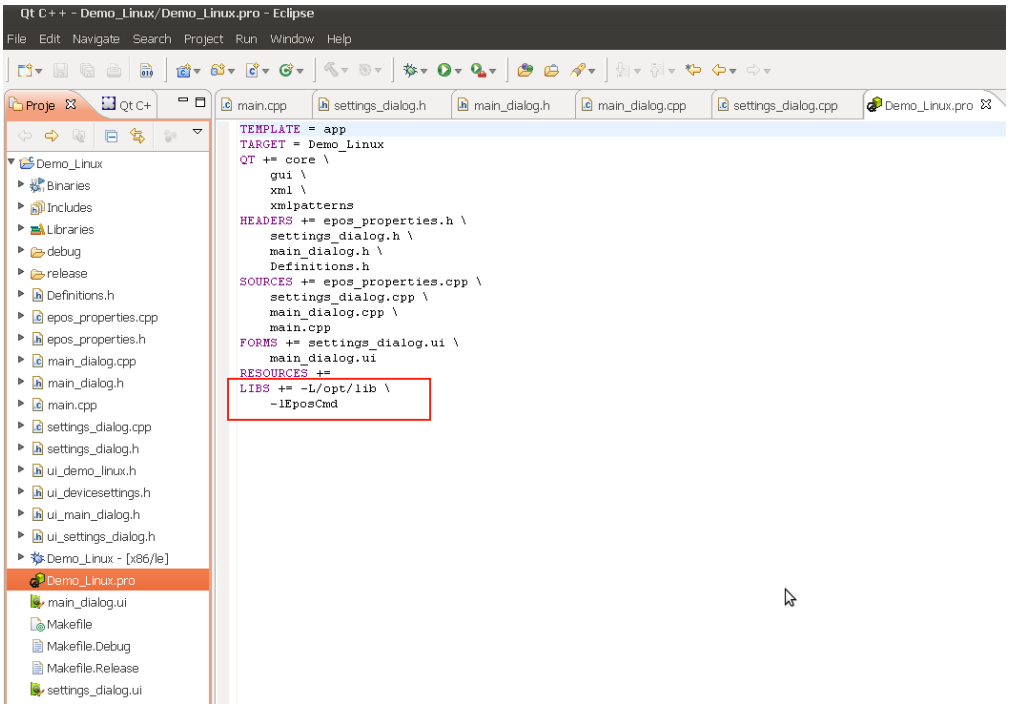

Figure 4: Edit QT project file

After these three steps you can execute directly all library functions in your own code.

6 Programming

The following chapter explains how the fundamental programming looks like. For several programming languages we deliver an example program.

6.1 Fundamental Program Flow

To configure the communication with the EPOS correctly, you have to execute an initialisation function before the first communication command. The exact program flow looks like this:

Initialisation procedures
These functions have to be executed at the beginning of the program.

Functions	Description
VCS_OpenDevice (...,...,...,...,…)	Initialisation of the port with the user data. For some information about the interface settings use the help functions.
VCS_SetProtocolStackSettings (…,…,…)	Initialisation of the new baud rate and timeout.
VCS_ClearFault (…,…,…)	Deletes existing errors.

Help Functions
These functions are needed, if you do not know exactly how your interface is configured.

Functions	Description
VCS_GetDeviceNameSelection (...,...,...,...,...,...)	This function returns all available 'DeviceNames' for the function "VCS_OpenDevice".
VCS_GetProtocolStackName-Selection (...,...,...,...,...,...)	This function returns all available 'ProtocolStackNames' for the function "VCS_OpenDevice".
VCS_GetInterfaceName-Selection (…,…,...,...,...,...)	This function returns all available 'InterfaceNames' for the function "VCS_OpenDevice".
VCS_GetPortNameSelection (…,…,...,...,...,...)	This function returns all available 'PortNames' for the function "VCS_OpenDevice".

Communication with EPOS
Choose any of the EPOS commands.

Functions	Description
VCS_OperationMode (…,…,…,…)	Set the operation mode (Position Mode, Profile Position Mode, Current Mode, …).
VCS_GetEncoderParameter (...,...,...,...)	Read all encoder parameters.
etc.	

Closing procedures
Before closing the program you have to release the port.

Functions	Description
VCS_CloseDevice (…,...)	Release the opened port.
VCS_CloseAllDevices (...)	Release all opened ports.

How the interfaces are exactly defined, is described in the document "EPOS Command Library"!

6.2 Eclipse C++/QT Example

Set the control parameters before you start this example program (e.g. motor and regulator parameters). Use the windows graphical user interface (EPOS Studio) for this configuration.

The example ‘Demo_Linux‘ is the QT (version 4.7, http://qt.digia.com) based dialog application . The application shows you, how the communication with the EPOS has to be configured.

First a configuration dialog is opened where you adjust your communication settings.

Figure 5: QT Demo communication settings

At the beginning the EPOS is set into ‘Profile Position Mode’. The whole initialisation is programmed in the member function “openDevice()“ of the class 'main_dialog‘. The opened port is released at the end in the function “closeDevice()“.

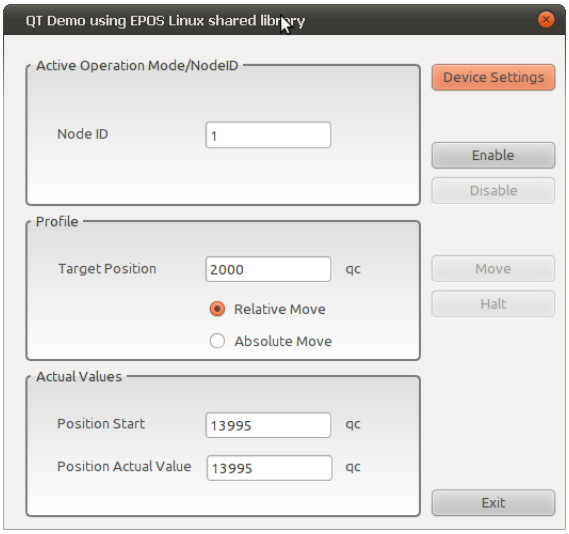

Figure 6: QT Demo main window

Clicking on the buttons you can execute the EPOS commands:

- “VCS_SetEnableState“,
- “VCS_SetDisableState“,
- “VCS_MoveToPosition”
- “VCS_HaltPositionMovement“.

The function “VCS_MoveToPosition” can be used as absolute or relative positioning. Clicking the button “Device Settings” you can change your communication settings.

A timer is triggering a periodical update of the state and the actual position. Every 100ms the function “UpdateView()“ is executed.