libxidcusb

1.4.1

Generated by Doxygen 1.7.1

Sun Dec 26 2010 19:58:22

Contents

Chapter 1

XIDC USB

Chapter 2

Introduction

2.1 About

Congratulations on choosing XIDCUSB multi-platform programing library! This document contain all information about XIDCUSB library. It utilizes well known FTDI virtual COM-port interface, so you can use it on Windows Vista, Windows XP, Windows Server 2003, Windows 2000, Windows ME, Windows 98, Linux, Mac OS X, etc., everything where free FTDI Virtual COM Port Driver installed. XIDCUSB multi-platform programing library can control up to 127 devices per computer. It supports plug/unplug on the fly. One program can control one device. Multiple processes (programs) that control one device simultaneously are not allowed.

2.2 System requirements

2.2.1 For rebuilding library

On Windows:

- Windows 2000 or later, 64-bit system (if compiling both arhitectures) or 32-bit system.
- Microsoft Visual C++ 2005 or later

On UNIX:

- 64- or 32-bit system
- gcc 4 or later
- common autotools: autoconf, autoheader, automake, autoreconf, libtool
- doxygen

On Mac OS X:

- gcc 4 or later
- XCode 3

2.2.2 For using library

On Windows:

- Windows 2000 or later, 64-bit system (if compiling both arhitectures) or 32-bit system.
- Microsoft Visual C++ 2005 or later (optional)
- mingw (optional)

On UNIX:

- 64- or 32-bit system
- gcc 4 or later
- debian or red hat based system with 2.6 kernel that could install DEB- or RPM-packages
- make

On Mac OS X:

- gcc 4 or later
- XCode 3

2.3 Revision history

2.3.1 version 1.4.1

sun, 26 dec 2010 02:00:00 +0400

- version bump
- feature #425: linux version doesn't query ttyACM ttys

2.3.2 version 1.4.0

wed, 24 nov 2010 02:00:00 +0400

- version bump
- feature #358: Protocol v12 support

2.3.3 version 1.3.0

sat, 16 oct 2010 02:00:00 +0400

- version bump
- feature #94: Visual Basic support

2.3.4 version 1.2.0

wed, 29 sep 2010 06:00:00 +0400

- version bump
- bug #286: hang while accessing two different devices

2.3.5 version 1.1.1r

thu, 22 jul 2010 15:00:00 +0400

- feature #294: documentation on russian

2.3.6 version 1.1.1

fri, 02 apr 2010 15:00:00 +0400

- version bump
- bug #267: firmware update doesn't work
- feature #264: development kit

2.3.7 version 1.1.0

fri, 19 feb 2010 15:00:00 +0400

- version bump
- bug #34: COM port
- bug #258: uint8_t, uint32_t
- feature #125: device detection protocol
- feature #226: linux device detection
- feature #237: update documentation
- feature #261: change commands S005, G005

2.3.8 version 1.0.0

sun, 12 dec 2009 21:00:00 +0400

- version bump
- bug #215: check crc at answers
- feature #91: How to use with C#
- feature #92: How to use with C++ Builder
- feature #93: How to use with Delphi
- feature #173: new command G1

2.3.9 version 0.9.0

sat, 15 aug 2009 21:00:00 +0400

- version bump
- Bug #24: Bugs with access from two applications
- Bug #56: Wrong data in cdc_pwmlevel
- Bug #76: Misspell in settings_t::flags
- Feature #27: Performance problems
- Feature #98: Documentation update
- Feature #99: Test program update

2.3.10 version 0.8.0

thu, 23 jul 2009 18:00:00 +0400

- version bump
- bug #11: support dll building with def-files
- bug #41: ttl_setup_outset can't be unset
- bug #42: engine_limit_curr, engine_limit_volt
- bug #54: open_device() returns -1 on second open, win32 = feature #21: mingw support

2.3.11 version 0.7.0

wed, 08 jul 2009 06:00:00 +0400

- version bump
- feature #39: Add firmware upload
- bug #33: Flags TTL_SYNCIN_SINGLE_SHIFT and TTL_SYNCIN_MULTI_SHIFT
- bug #35: write_key
- bug #36: get_analog_data

2.3.12 version 0.6.0

tue, 30 jun 2009 17:00:00 +0400

- version bump
- bug #12: mistyping at pid_settings_t: kprpm, kiprm, kdprm
- bug #13: error at set_pwm_freq
- bug #14: initialization error

- bug #16: error writing synchonization params
- bug #17: error at param ttl_setup_front
- bug #25: error reading calibration coefficients
- feature #32: add 1234, calb, reset
- add writekey function

2.3.13 Version 0.5.0

Fri, 21 May 2009 01:12:54 +0400

- Version bump
- Updated documentation
- Fix memory problem
- Improve compatibility with C++ on windows

2.3.14 Version 0.4.0

Fri, 06 May 2009 11:37:29 +0400

- Version bump
- Add documentation

2.3.15 Version 0.3.0

Fri, 24 Apr 2009 11:17:29 +0400

- Version bump
- Renamed
- Fixed unix build
- Add force unlocking semaphores

2.3.16 Version 0.2.0

Sun, 05 Apr 2009 19:02:43 +0400

- Version bump
- Added deb and rpm packages building
- Mac OS X library is private framework now
- Add version numbers in win32 version

2.3.17 Version 0.1.0

Thu, 02 Apr 2009 00:02:44 +0400

- Initial release. Basic functionality.

Chapter 3

How to rebuild library

3.1 Building on generic UNIX

Invoke following commands:

```
$ ./autogen.sh
$ ./configure
```

For 32-bit system:

```
$ make
```

For 64-bit system:

```
$ CFLAGS="-m64" make
```

Install into /usr/local:

```
$ sudo make install
```

Also generate docs:

```
$ make doxygen-doc
```

Compiled library - /usr/local/lib/libxidcusb.so, header file - /usr/local/include/xidcusb.h

3.1.1 Packaging deb

Requirement: 64-bit debian system, gcc, autotools, dpkg-dev.

First of all, increase version number with:

```
$ ./version.sh 0.42.0
```

Run packaging script:

```
$ ./builddeb.sh 0.42.0
```

3.1.2 Packaging rpm

Requirement: 64-bit RPM-based system (Fedora, Red Hat, SUSE), gcc, autotools.

First of all, increase version number with:

```
$ ./version.sh 0.42.0
```

Run packaging script:

```
$ ./buildrpm.sh 0.42.0
```

3.2 Buliding on Darwin

Invoke following commands:

```
$ ./autogen.sh
$ ./configure
```

For 32-bit system:

```
$ make
```

For 64-bit system:

```
$ make CFLAGS="-arch x86_64" LDFLAGS="-arch x86_64"
```

Install into /usr/local:

```
$ sudo make install
```

Compiled library - /usr/local/lib/libxidcusb.dylib, header file - /usr/local/include/xidcusb.h

Note that only 32- or 64-bit library can be installed at the same time.

3.3 Buliding framework on Mac OS X

Buillding Mac OS X Framework Library is supported.
Invoke command:

```
$ xcodebuild
```

or open and build libxidusb.xcodeproj with XCode 3.1

Compiled framework libxidcusb.framework can be found at 'dist/<version>/macosx'

3.4 Buliding on Windows

Ensure that version number is already increased with version.sh script.

Open and build with Visual Studio 2005/2008 solution libxidcusb.sln.

Compiled dll libxidcusb.dll can be found at 'Release/Win32' or 'Release/x64'

Alternatively, you can run from Visual Studio command prompt batch-file build.bat. It will build all 32- and 64-bit libraries and push them to 'dist/<version>/win32' or 'dist/<version>/x64' directories.

3.5 Source code access

XIDCUSB source codes are given under special request.

Chapter 4

How to use with...

Library usage can be examinated from test application testapp. Non-C languages are supported because library supports stdcall calling convention and so can be used with a variety of languages.

C test project is located at 'testapp' directory, C# test project - at 'testcs', VB.NET - 'testvbnet', Delphi 6 - 'testdelphi'. Libraries, headers and other required files are located at 'win32'/'win64','macosx' directories.

4.1 Visual C++

Testapp can be built using testapp.sln. It's needed to specify libxidcusb.lib in linker input. Library must be compiled with MS Visual C++ too, mingw-library isn't supported. Make sure that Microsoft Visual C++ Redistributable Package is installed.

NOTE: Example compiled with(MS Visual C++ 2008 SP1 and needs package 9.0.307291 (provided with SDK - vcredist_x86 or vcredist_x64)

Steps to build example:

- Build solution testapp.sln with target architecture
- Copy library libxidcusb.dll to Release∗ directory
- Launch testapp.exe in console

4.2 MinGW

MinGW is a port of GCC to win32 platform. It's required to install MinGW package.

MinGW-compiled testapp can be built with MS Visual C++ or mingw library.

```
$ mingw32-make -f Makefile.mingw all
```

Then copy library libxidcusb.dll to current directory and launch testapp.exe.

4.3 C++ Builder

First of all you should create C++ Builder-style import library. Visual C++ library is not compatible with BCB. Invoke:

```
$ implib libxidcusb.lib libxidcusb.lib
```

Then compile test application:

```
$ bcc32 -I..\libxidcusb\include -DWIN32 -DNDEBUG -D_WINDOWS testapp.c libxidcusb
     .lib
```

4.4 .NET

Wrapper assembly for libxidcusb.dll is xidcusbnet.dll. It is provided with two different architectures and depends on .NET 2.0.

Test .NET applications for Visual Studio 2008 is located at testcs (for C#) and testvbnet (for VB.NET) respectively. Just compile, place libxidcusb.dll near the output assemblies and run program.

4.5 Delphi

Wrapper for libxidcusb.dll is provided as unit include/xidcusb.pas

Console test application for is located at testdelphi. Tested with Delphi 6 and only 32-bit version.

Just compile, place DLL near the executable and run program.

4.6 XCode

Testapp should be built with XCode project testapp.xcodeproj. Use Release configuration. Library is a MAC OS X framework, and at example application it's bundled inside Testapp.app

Then launch application Testapp.app and check activity output in Console.app.

4.7 GCC

Make sure that libxidcusb (RPM or DEB package) is installed at your system. Installation of package should be performed with package manager of operating system.

Testapp should be built with:

```
$ make all
```

Then launch application as:

```
$ ./testapp
```

Chapter 5

Programming interfaces

Reference: xidcusb.h

Chapter 6

Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 7

File Index

7.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 8

Data Structure Documentation

8.1 analog_data_t Struct Reference

Analog data.

`#include <xidcusb.h>`

Data Fields

- int adc_va1

  "Voltage on pin 1 winding A" raw data from ADC

- int adc_va2

  "Voltage on pin 2 winding A" raw data from ADC

- int adc_vb1

  "Voltage on pin 1 winding B" raw data from ADC

- int adc_vext

  "External voltage" raw data from ADC

- int adc_vsup

  "Voltage on the top of MOSFET full brige" raw data from ADC

- int adc_acur

  "Winding A current" raw data from ADC

- int adc_bcur

  "Winding B current" raw data from ADC

- int adc_fullcur

  "Full current" raw data from ADC

- int adc_pot

  Voltage on "Potentiometer", raw data from ADC.

- int adc_tempkey

  Voltage from temperature sensor, raw data from ADC.

- int cdc_va1

  "Voltage on pin 1 winding A" calibrated data

- int cdc_va2

  "Voltage on pin 2 winding A" calibrated data

- int cdc_vb1

  "Voltage on pin 1 winding B" calibrated data

- int cdc_vext

  "External voltage" calibrated data

- int cdc_vsup

  "Voltage on the top of MOSFET full brige" calibrated data

- int cdc_acur

  "Winding A current" calibrated data

- int cdc_bcur

  "Winding B current" calibrated data

- int cdc_fullcur

  "Full current" calibrated data

- int cdc_tempkey

  Temperature, calibrated data.

- int cdc_dutycycle

  Duty cycle of PWM.

8.1.1 Detailed Description

Analog data. This structure contains raw analog data from ADC embedded on board. These data used for device testing and deep recalibraton by manufacturer only

See also

get_analog_data

The documentation for this struct was generated from the following file:

- include/xidcusb.h

8.2 calibration_settings_t Struct Reference

Calibration settings.

`#include <xidcusb.h>`

Data Fields

- int amp_vout1

  Scale factor for "Voltage out1".

- int amp_vout2

  Scale factor for "Voltage out2".

- int amp_vext

  Scale factor for "Supply voltage".

- int amp_vkey

  Scale factor for raw voltage on the top of MOSFET full brige.

- int amp_engcur

  Scale factor for "Engine current".

- int amp_tempkey

  Scale factor for raw data from temperature sensor.

- int off_vout1

  Offset factor for "Voltage out1".

- int off_vout2

  Offset factor for "Voltage out2".

- int off_vext

  Offset factor for "Supply voltage".

- int off_vkey

  Offset factor for raw voltage on the top of MOSFET full brige.

- int off_engcur

  Offset factor for "Engine current".

- int off_tempkey

  Offset factor for raw data from temperature sensor.

8.2.1 Detailed Description

Calibration settings. This structure contains calibration settings for all analog data from ADC embedded on board. Range: -32767..32767. All boards are supplied with full set of calibration data on controller's flash memory. There is no necessity to change it during normal operation. These data have to be changed after special device recalibraton only. If you want to recalibrate board please consult with the manufacturer. Wrong calibration settings lead to device malfunction, can lead to irreversible damage of board

See also

get_calibration_coeffs
set_calibration_coeffs

The documentation for this struct was generated from the following file:

- include/xidcusb.h

8.3 chart_data_t Struct Reference

Additional device state.

`#include <xidcusb.h>`

Data Fields

- int winding_voltage_a

  Winding A voltage.

- int winding_voltage_b

  Winding B voltage.

- int winding_voltage_c

  Winding C voltage.

- int winding_current_a

  Winding A current.

- int winding_current_b

  Winding B current.

- int winding_current_c

  Winding C current.

- int current_temp

  Temperature of MOSFET keys.

8.3.1 Detailed Description

Additional device state. This structure contains additional values such as winding's voltages, currents and temperature.

See also

get_status

The documentation for this struct was generated from the following file:

- include/xidcusb.h

8.4 engine_settings_t Struct Reference

Engine settings.

`#include <xidcusb.h>`

Data Fields

- unsigned int nom_voltage
  Rated voltage.
- unsigned int nom_input
  Rated current.
- unsigned int nom_rpm
  Rated RPM.
- unsigned int flags
  Set of flags specify motor shaft movement algorithm and list of limitations.
- int antiplay
  Number of pulses for backlash (play) compensation procedure.
- int microstep_mode
  Settings of microstep mode.
- int type
  Defines motor type.

8.4.1 Detailed Description

Engine settings. This structure contains useful motor settings. These settings specify motor shaft movement algorithm, list of limitations and rated characteristics. All boards are supplied with standart set of engine setting on controller's flash memory. Please load new engine settings when you change motor, encoder, positioner etc. Please note that wrong engine settings lead to device malfunction, can lead to irreversible damage of board

See also

set_engine_settings
get_engine_settings

8.4.2 Field Documentation

8.4.2.1 int antiplay

Number of pulses for backlash (play) compensation procedure.

Used if ENGINE_ANTIPLAY flag is set. Range: 1..65535

8.4.2.2 unsigned int nom_input

Rated current.

Controller will keep current consumed by motor below this value if ENGINE_LIMIT_CURR flag is set. Range: 1..65535

8.4.2.3 unsigned int nom_rpm

Rated RPM.

Controller will keep motor shaft RPM below this value if ENGINE_LIMIT_RPM flag is set. Range: 1..65535

8.4.2.4 unsigned int nom_voltage

Rated voltage.

Controller will keep the voltage drop on motor below this value if ENGINE_LIMIT_VOLT flag is set. Range: 1..65535

The documentation for this struct was generated from the following file:

- include/xidcusb.h

8.5 home_settings_t Struct Reference

Position calibration settings.

```
#include <xidcusb.h>
```

Data Fields

- int fast_home
  Speed used for fast motion.
- int slow_home
  Speed used for slow motion.

- unsigned int delta

  Distance from break point.

- unsigned int flags

  Set of flags specify direction and stopping conditions.

8.5.1 Detailed Description

Position calibration settings. This structure contains settings used in position calibrating. It specify behaviour of calibrating position.

See also

get_home_settings
set_home_settings
command_home

8.5.2 Field Documentation

8.5.2.1 unsigned int delta

Distance from break point.

Range: 0..2147483647

8.5.2.2 int fast_home

Speed used for fast motion.

Range: 1..65535.

8.5.2.3 int slow_home

Speed used for slow motion.

Range: 1..65535.

The documentation for this struct was generated from the following file:

- include/xidcusb.h

8.6 pid_settings_t Struct Reference

PID settings.

`#include <xidcusb.h>`

Data Fields

- unsigned int kpi

Proportional gain for current PID routine.

- unsigned int kii

  Integral gain for current PID routine.

- unsigned int kdi

  Differential gain for current PID routine.

- unsigned int kpu

  Proportional gain for voltage PID routine.

- unsigned int kiu

  Integral gain for voltage PID routine.

- unsigned int kdu

  Differential gain for voltage PID routine.

- unsigned int kprpm

  Proportional gain for speed PID routine.

- unsigned int kirpm

  Integral gain for speed PID routine.

- unsigned int kdrpm

  Differential gain for speed PID routine.

- unsigned int kppos

  Proportional gain for position PID routine.

- unsigned int kipos

  Integral gain for position PID routine.

- unsigned int kdpos

  Differential gain for position PID routine.

- unsigned int kpe1

  Proportional gain for ext1 PID routine.

- unsigned int kie1

  Integral gain for ext1 PID routine.

- unsigned int kde1

  Differential gain for ext1 PID routine.

- unsigned int kpe2

  Proportional gain for ext2 PID routine.

- unsigned int kie2

  Integral gain for ext2 PID routine.

- unsigned int kde2

  Differential gain for ext2 PID routine.

- unsigned int kpe3

  Proportional gain for ext3 PID routine.

- unsigned int kie3

  Integral gain for ext3 PID routine.

- unsigned int kde3

  Differential gain for ext3 PID routine.

8.6.1 Detailed Description

PID settings. This structure contains factors for PID routines. Range: 0..65535. It specify behaviour of PID routines for position, speed, voltage and current. These factors are slightly different for different positioners. All boards are supplied with standart set of PID setting on controller's flash memory. Please load new PID settings when you change positioner. Please note that wrong PID settings lead to device malfunction

See also

set_pid_settings
get_pid_settings

The documentation for this struct was generated from the following file:

- include/xidcusb.h

8.7 state_t Struct Reference

Device state.

`#include <xidcusb.h>`

Data Fields

- DevState device_state

  Device state.

- PwmState pwm_state

  PWM state, PWM module drive MOSFET full brige that control current through the motor.

- unsigned int flags

  Set of flags specify motor shaft movement algorithm and list of limitations.

- unsigned int current_position

  Motor shaft position.

- int current_rpm
  Motor shaft speed, RPM.
- int current_input
  Engine current.
- int current_engine_voltage
  Current average engine voltage.
- int current_power_voltage
  Power supply voltage.
- int current_temp
  MOSFET full brige temperature.

8.7.1 Detailed Description

Device state. Useful structure that contains current controller status, including speed, position and boolean flags

See also

get_status

The documentation for this struct was generated from the following file:

- include/xidcusb.h

8.8 sync_settings_t Struct Reference

Synchronization settings.

`#include <xidcusb.h>`

Data Fields

- unsigned int setup_flags
  Generic setup flags.
- unsigned int syncin_flags
  Input synchronization flags.
- unsigned int syncout_flags
  Output synchronization flags.
- SyncPinMode sync_pin_mode
  Synchronization pin mode.

- unsigned int impulse_time

  Output synchronization pulse duration or input synchronization pulse dead time.

- unsigned int impulse_period

  This value specify number of encoder pulses between two output synchronization pulses when TTL_SYNCOUT_ONPERIOD is set.

8.8.1 Detailed Description

Synchronization settings. This structure contains all synchronization settings, modes, periods and flags. It specify behaviour of synchronization In/Out. All boards are supplied with standart set of these settings

See also

get_sync_settings
set_sync_settings

8.8.2 Field Documentation

8.8.2.1 unsigned int impulse_period

This value specify number of encoder pulses between two output synchronization pulses when TTL_SYNCOUT_ONPERIOD is set.

Range: 0..65535

8.8.2.2 unsigned int impulse_time

Output synchronization pulse duration or input synchronization pulse dead time.

Range: 0..65535

The documentation for this struct was generated from the following file:

- include/xidcusb.h

Chapter 9

File Documentation

9.1 include/xidcusb.h File Reference

Header file for XIDCUSB library.

Data Structures

- struct state_t

  Device state.

- struct chart_data_t

  Additional device state.

- struct calibration_settings_t

  Calibration settings.

- struct engine_settings_t

  Engine settings.

- struct pid_settings_t

  PID settings.

- struct sync_settings_t

  Synchronization settings.

- struct home_settings_t

  Position calibration settings.

- struct analog_data_t

  Analog data.

Defines

- #define DCCONTROL_API __attribute__((visibility("default")))

  Library import macros.

- #define device_undefined -1

  Handle specified undefined device.

Result statuses

- #define result_ok 0

  success

- #define result_error -1

  generic error

- #define result_not_implemented -2

  function is not implemented

- #define result_dataerror -3

  device reports wrong request or response

- #define result_nodevice -4

  device is lost

Status flags

Controller flags returned by device query.

Contains boolean part of controller status. May be combined with bitwise OR

See also

state_t::flags
get_status

- #define STATE_RIGHT_EDGE 0x0001

  Engine stucks at the right edge.

- #define STATE_LEFT_EDGE 0x0002

  Engine stucks at the left edge.

- #define STATE_EXT_POWER 0x0004

  External power is used.

- #define STATE_ERRC 0x0008

  Command error encountered.

- #define STATE_ERRD 0x0010

  Data error encountered.

- #define STATE_POWER_DISABLE 0x0020

  FTDI reports "Power disable".

- #define STATE_TTLIO_PINOUT 0x0040
  Synchronization In/Out works as Out, if flag set; otherwise works as In.
- #define STATE_TTLIO_LEVEL 0x0080
  State of synchronization In/Out pin.
- #define STATE_BUTTON_RIGHT 0x0100
  Button "right" state (1 if pressed).
- #define STATE_BUTTON_LEFT 0x0200
  Button "left" state (1 if pressed).
- #define STATE_OVERHEAT 0x2000
  Temperature sensor show value that exceed maximum temperature rating less then 1 ms.
- #define STATE_OVERLOAD_VOLTAGE 0x4000
  Voltage overload encountered.
- #define STATE_OVERLOAD_CURRENT 0x8000
  Current overlaod encountered.

Flags of engine settings

Specify motor shaft movement algorithm and list of limitations.

Flags returned by query of engine settings. May be combined with bitwise OR

See also

engine_settings_t::flags
set_engine_settings
get_engine_settings

- #define ENGINE_REVERSE 0x0001
  Reverse flag.
- #define ENGINE_FINISHING 0x0002
  Fine tuneup flag.
- #define ENGINE_MAX_SPEED 0x0004
  Max speed flag.
- #define ENGINE_ANTIPLAY 0x0008
  Play compensation flag.
- #define ENGINE_ACCEL_ON 0x0010
  Acceleration enable flag.
- #define ENGINE_LIMIT_VOLT 0x0020
  Maxumum motor voltage limit enable flag.
- #define ENGINE_LIMIT_CURR 0x0040
  Maxumum motor current limit enable flag.

- #define ENGINE_LIMIT_RPM 0x0080
  Maxumum motor speed limit enable flag.
- #define ENGINE_HOLD 0x0100
  Hold motor shaft when motor is stopped if this flag set.
- #define ENGINE_DYNAMIC_HOLD 0x0200
  Holding motor shaft dynamically (power supply is applied to motor always), if set; when shaft position is different from specified, if not set.
- #define ENGINE_ONLY_FINISHING 0x0400
  Move in fine tuneup mode only if this flag set.

Flags of microstep mode

Specify settings of microstep mode.

Used with step motors. Flags returned by query of engine settings. May be combined with bitwise OR

See also

engine_settings_t::flags
set_engine_settings
get_engine_settings

- #define MICROSTEP_MODE_FULL 0x01
  Full step mode.
- #define MICROSTEP_MODE_FRAC_2 0x02
  1/2 step mode.
- #define MICROSTEP_MODE_FRAC_4 0x04
  1/4 step mode.
- #define MICROSTEP_MODE_FRAC_8 0x08
  1/8 step mode.
- #define MICROSTEP_MODE_FRAC_16 0x10
  1/16 step mode.
- #define MICROSTEP_MODE_FRAC_32 0x20
  1/32 step mode.
- #define MICROSTEP_MODE_FRAC_64 0x40
  1/64 step mode.
- #define MICROSTEP_MODE_FRAC_128 0x80
  1/128 step mode.

Flags of engine type

Specify motor type.

Flags returned by query of engine settings.

See also

engine_settings_t::flags
set_engine_settings
get_engine_settings

- #define ENGINE_TYPE_DC 0x01
  DC motor.
- #define ENGINE_TYPE_STEP 0x02
  Step motor.
- #define ENGINE_TYPE_BRUSHLESS 0x04
  Brushless motor.

- #define FEEDBACK_POTENTIOMETER 0x00
  Describes feedback type.
- #define FEEDBACK_ENCODER 0x01
  feedback by encoder.
- #define FEEDBACK_ENCODERDIFF 0x02
  feedback by encoder with differential input.
- #define FEEDBACK_ENCODERHALL 0x04
  feedback by Hall detector.
- #define FEEDBACK_EMF 0x08
  feedback by EMF.
- #define FEEDBACK_NONE 0x10
  feedback is absent.

Flags of synchronization In/Out setup

See also

sync_settings_t::setup_flags
get_sync_settings
set_sync_settings

- #define TTL_SETUP_DIROUT 0x0001
  Synchronization In/Out works as Out, if flag set; otherwise works as In.
- #define TTL_SETUP_OUTSET 0x0002
  Value on Synchronization Out, used if direction is out only.
- #define TTL_SETUP_FRONT 0x0010
  Synchronization In mode.

Flags for synchronization input setup

See also

sync_settings_t::syncin_flags
get_sync_settings
set_sync_settings

- #define TTL_SYNCIN_ONMOVE 0x0001
  Infinite moving in TTL_SYNCIN_DIRRIGHT direction, if set; moving to given position or given shift, if not set.
- #define TTL_SYNCIN_MULTI_SHIFT 0x0004
  Response on every incoming pulse, if set; one time response, if not set.
- #define TTL_SYNCIN_DIRRIGHT 0x0008
  Movement direction to the right, if set.

Flags of synchronization output

See also

sync_settings_t::syncout_flags
get_sync_settings
set_sync_settings

- #define TTL_SYNCOUT_ONSTART 0x0001
  Generate synchronization pulse when moving started.
- #define TTL_SYNCOUT_ONSTOP 0x0002
  Generate synchronization pulse when moving stopped.
- #define TTL_SYNCOUT_ONPERIOD 0x0004
  Generate synchronization pulse every impulse_period encoder pulses.

Border flags

Specify types of borders and motor behaviour on borders.

May be combined with bitwise OR

See also

get_edges_settings
set_edges_settings

- #define BORDER_IS_ENCODER 0x0001
  Borders are fixed by predetermined encoder values, if set; borders position on limit switches, if not set.
- #define BORDER_STOP_LEFT 0x0002
  Motor should stop on left border.
- #define BORDER_STOP_RIGHT 0x0004
  Motor should stop on right border.

Limit switches flags

Specify electrical behaviour of limit switches like order and pulled positions.

May be combined with bitwise OR

See also

get_edges_settings
set_edges_settings

- #define ENDER_DIRECT 0x0001
  First limit switch on the right side, if set; otherwise on the left side.
- #define ENDER_ON1 0x0002
  First limit switch pulled position is closed, if set; otherwise pulled position is opened.
- #define ENDER_ON2 0x0004
  Second limit switch pulled position is closed, if set; otherwise pulled position is opened.

Home settings flags

Specify behaviour for home command.

May be combined with bitwise OR

See also

get_home_settings
set_home_settings
command_home

- #define HOME_DIR_FAST 0x01
  Flag defines direction of 1st motion after execute home command.
- #define HOME_STOP_FAST 0x02
  Fast motion stops by synchronization input, if set; otherwise by limit switch.
- #define HOME_DIR_SLOW 0x04
  Flag defines direction of 2nd motion.
- #define HOME_REV_EN 0x08
  Revolution sensor is used, if set; otherwise this action passed.

Typedefs

- typedef int device_t
  Type describes device identifier.
- typedef int result_t
  Type specifies result of any XIDC operation.

Enumerations

- enum DevState {
  DevStateOff, DevStateStop, DevStateMoving, DevStateTune,
  DevStateHoming, DevStateCalibr = 0x0C, DevStateAlarm = 0x0A }

Describes motor state.

- enum PwmState {
  PwmStateOff, PwmStateBrak, PwmStateRunfwd, PwmStateRunbck,
  PwmStateInvbrak }
  Describes PWM state.

- enum SyncPinMode { PinGPIO = 0x00, PinSyncIn = 0x01, PinSyncOut = 0x02 }
  Describes synchronization pin mode.

Functions

Boards and drivers control

Functions for searching and opening/closing devices

- device_t DCCONTROL_API open_device (const char *name)
  Open a device with OS name name and return identifier of the device which can be used in XIDC calls.

- result_t DCCONTROL_API close_device (device_t id)
  Close specified device.

- result_t DCCONTROL_API probe_device (const char *name)
  Check if a device with OS name name is XIMC device.

- result_t DCCONTROL_API enumerate_devices (int *name_count, char ***names, int probe_-devices)
  Enumerate all devices that looks like XIDC valid.

- result_t DCCONTROL_API free_enumerate_devices (int name_count, char **names)
  Free memory returned by enumerate_devices.

- result_t DCCONTROL_API init_device (device_t id)
  Init device.

- result_t DCCONTROL_API deinit_device (device_t id)
  Deinit device.

- result_t DCCONTROL_API get_device_information (device_t id, char *manufacturer, char *manufacturer_id, char *product_description)
  Return device information.

- result_t DCCONTROL_API reset_locks ()
  Reset library locks in a case of deadlock.

Engine control

Main functions for controlling engine movement

- result_t DCCONTROL_API command_move (device_t id, unsigned int pos)
  Move engine to position pos.

- result_t DCCONTROL_API command_movr (device_t id, int offset)

Move engine to the distance offset relatively to initial position.

- result_t DCCONTROL_API command_left (device_t id)
  Start continous moving to the left.
- result_t DCCONTROL_API command_right (device_t id)
  Start continous moving to the right.
- result_t DCCONTROL_API command_stop (device_t id)
  Stop engine.
- result_t DCCONTROL_API command_zero (device_t id)
  Reset current position to zero.
- result_t DCCONTROL_API command_home (device_t id)
  Position calibrating Move in direction, defined by HOME_DIR_FAST (right, if set; otherwise - left) with speed home_settings_t::fast_home.
- result_t DCCONTROL_API get_status (device_t id, state_t *state)
  Return device state.

Engine setup

Functions for adjusting engine read/write almost all controller settings

- result_t DCCONTROL_API get_pwm_freq (device_t id, unsigned int *freq)
  Read pwm frequency.
- result_t DCCONTROL_API set_pwm_freq (device_t id, unsigned int freq)
  Set pwm frequency.
- result_t DCCONTROL_API get_feedback_settings (device_t id, unsigned int *ips, unsigned int *feedback_flags)
  Read feedback settings.
- result_t DCCONTROL_API set_feedback_settings (device_t id, unsigned int ips, unsigned int feedback_flags)
  Set feedback settings.
- result_t DCCONTROL_API get_move_settings (device_t id, unsigned int *rpm, unsigned int *accel, unsigned int *tuneup_threshold)
  Read specified motor shaft speed and acceleration.
- result_t DCCONTROL_API set_move_settings (device_t id, unsigned int rpm, unsigned int accel, unsigned int tuneup_threshold)
  Set specified for this and next motion motor shaft speed and acceleration.
- result_t DCCONTROL_API get_engine_settings (device_t id, engine_settings_t *engine_settings)
  Read engine settings.
- result_t DCCONTROL_API set_engine_settings (device_t id, const engine_settings_t *engine_settings)
  Set engine settings.

- result_t DCCONTROL_API get_secure_settings (device_t id, unsigned int *critical_curr, unsigned int *critical_voltage, unsigned int *critical_temp)
  Read engine voltage, current and temperature maxumum ratings.

- result_t DCCONTROL_API set_secure_settings (device_t id, unsigned int critical_curr, unsigned int critical_voltage, unsigned int critical_temp)
  Set engine voltage, current and temperature maxumum ratings.

- result_t DCCONTROL_API get_edges_settings (device_t id, int *border_flags, int *enders_flags, int *left, int *right)
  Read border and limit switches settings.

- result_t DCCONTROL_API set_edges_settings (device_t id, int border_flags, int enders_flags, int left, int right)
  Set border and limit switches settings.

- result_t DCCONTROL_API get_pid_settings (device_t id, pid_settings_t *pid_settings)
  Read PID settings.

- result_t DCCONTROL_API set_pid_settings (device_t id, const pid_settings_t *pid_settings)
  Set PID settings.

- result_t DCCONTROL_API get_sync_settings (device_t id, sync_settings_t *sync_settings)
  Read synchronization settings.

- result_t DCCONTROL_API set_sync_settings (device_t id, const sync_settings_t *sync_settings)
  Set synchronization settings.

- result_t DCCONTROL_API get_home_settings (device_t id, home_settings_t *home_settings)
  Read home settings.

- result_t DCCONTROL_API set_home_settings (device_t id, const home_settings_t home_settings)
  Set home settings.

- result_t DCCONTROL_API command_read_settings (device_t id)
  Read all settings from controller's flash memory to controller's RAM, replacing previous data in controller's RAM.

- result_t DCCONTROL_API command_save_settings (device_t id)
  Save all settings from controller's RAM to controller's flash memory, replacing previous data in controller's flash memory.

- result_t DCCONTROL_API get_serial_number (device_t id, uint32_t *serial)
  Read device serial number.

- result_t DCCONTROL_API get_firmware_version (device_t id, unsigned int *major, unsigned int *minor, unsigned int *release)
  Read controller's firmware version.

- result_t DCCONTROL_API get_analog_data (device_t id, analog_data_t *ad)
  Read analog data structure that contains raw analog data from ADC embedded on board.

- result_t DCCONTROL_API get_chart_data (device_t id, chart_data_t *chart_data)

Return device electrical parameters, useful for charts.

Service functions

Functions are intended for deep controller setup.

There is no necessity to use it during normal operation. These functions have to be used by skilled engineers only. If you want to use it please consult with the manufacturer. Wrong usage lead to device malfunction, can lead to irreversible damage of board

- result_t DCCONTROL_API get_calibration_coeffs (device_t id, calibration_settings_t *cs)
  Read calibration coefficients.
- result_t DCCONTROL_API set_calibration_coeffs (device_t id, const calibration_settings_t *cs)
  Set calibration coefficients.
- device_t DCCONTROL_API open_raw_device (const char *name)
  Open a device with OS name name and return identifier of the device which can be used in XIDC calls.
- result_t DCCONTROL_API set_serial_number (device_t id, uint32_t serial, uint32_t key)
  Write device serial number to controller's flash memory.
- result_t DCCONTROL_API has_firmware (device_t id, uint8_t *ret)
  Check for firmware on device.
- result_t DCCONTROL_API write_key (device_t id, uint32_t key)
  Write controller key.
- result_t DCCONTROL_API command_calibrate (device_t id)
  Calibrate controller.
- result_t DCCONTROL_API command_reset (device_t id)
  Reset controller.
- result_t DCCONTROL_API command_update_firmware (device_t id, const uint8_t *data, uint32_t data_size)
  Update firmware.

9.1.1 Detailed Description

Header file for XIDCUSB library.

9.1.2 Define Documentation

9.1.2.1 #define DCCONTROL_API __attribute__((visibility("default")))

Library import macros.

Macros allows to automatically import function from shared library. It automatically expands to dllimport on msvc when including header file

9.1.2.2 #define ENGINE_ACCEL_ON 0x0010

Acceleration enable flag.

If it set, motion begins with acceleration and ends with deceleration

9.1.2.3 #define ENGINE_ANTIPLAY 0x0008

Play compensation flag.

If it set, engine makes backlash (play) compensation procedure and reach the predetermined position accurately on low speed

9.1.2.4 #define ENGINE_FINISHING 0x0002

Fine tuneup flag.

If it set, engine makes special tuneup procedure and reach the predetermined end position accurately on low speed

9.1.2.5 #define ENGINE_MAX_SPEED 0x0004

Max speed flag.

If it is set, engine uses maxumum speed achievable with the present engine settings as nominal speed

9.1.2.6 #define ENGINE_ONLY_FINISHING 0x0400

Move in fine tuneup mode only if this flag set.

Do not use this flag in normal operation mode. It used for PID tuning only

9.1.2.7 #define ENGINE_REVERSE 0x0001

Reverse flag.

It determines motor shaft rotation direction that corresponds to feedback counts increasing. If not set (default), motor shaft rotation direction under positive voltage corresponds to feedback counts increasing and vice versa. Change it if you see that positive directions on motor and feedback are opposite

9.1.2.8 #define FEEDBACK_POTENTIOMETER 0x00

Describes feedback type.

See also

set_feedback_settings
get_feedback_settings feedback by potentiometer.

9.1.2.9 #define HOME_DIR_FAST 0x01

Flag defines direction of 1st motion after execute home command.

Direction is right, if set; otherwise left.

9.1.2.10 #define HOME_DIR_SLOW 0x04

Flag defines direction of 2nd motion.

Direction is right, if set; otherwise left.

9.1.2.11 #define STATE_OVERLOAD_CURRENT 0x8000

Current overlaod encountered.

Motor current exceed maximum rating less then 1 ms

9.1.2.12 #define STATE_OVERLOAD_VOLTAGE 0x4000

Voltage overload encountered.

Voltage drop on motor exceed maximum rating less then 1 ms

9.1.2.13 #define TTL_SETUP_FRONT 0x0010

Synchronization In mode.

Response on front, if set. Response on cut, if not set

9.1.2.14 #define TTL_SYNCIN_DIRRIGHT 0x0008

Movement direction to the right, if set.

Used when TTL_SYNCIN_ONMOVE set

9.1.3 Enumeration Type Documentation

9.1.3.1 enum DevState

Describes motor state.

Enumerator:

DevStateOff motor is turned off.
DevStateStop motor is stopped.
DevStateMoving motor is moving.
DevStateTune motor is tuning.
DevStateHoming motor is going to home position.
DevStateCalibr motor is calibrating.
DevStateAlarm engine in alarm state.

9.1.3.2 enum PwmState

Describes PWM state.

Pulse-width modulation (PWM) module drive MOSFET full brige that control current through the motor

Enumerator:

PwmStateOff PWM is off, motor winding is cut off power.

PwmStateBrak braking mode, motor winding is closed.

PwmStateRunfwd motor running forward.

PwmStateRunbck motor running backward.

PwmStateInvbrak inverted braking mode.

9.1.3.3 enum SyncPinMode

Describes synchronization pin mode.

Enumerator:

PinGPIO GPIO.

PinSyncIn sync in.

PinSyncOut sync out.

9.1.4 Function Documentation

9.1.4.1 result_t DCCONTROL_API close_device (device_t *id*)

Close specified device.

Parameters

id an identifier of device

9.1.4.2 result_t DCCONTROL_API command_calibrate (device_t *id*)

Calibrate controller.

Can be used by manufacturer only

Parameters

id an identifier of device

9.1.4.3 result_t DCCONTROL_API command_home (device_t *id*)

Position calibrating Move in direction, defined by HOME_DIR_FAST (right, if set; otherwise - left) with speed home_settings_t::fast_home.

If HOME_STOP_FAST is set, motor stops by synchronization input activity; otherwise - by limit switch. Shift in reverse direction to home_settings_t::home_delta with speed home_settings_t::slow_home. Move

in direction, defined by HOME_DIR_SLOW (right, if set; otherwise - left) with speed home_settings_t::slow_home. If HOME_STOP_SLOW is set, motor stops by limit switch; otherwise - by synchronization input activity.

Parameters

id an identifier of device

See also

home_settings_t
get_home_settings
set_home_settings

9.1.4.4 result_t DCCONTROL_API command_left (device_t *id*)

Start continous moving to the left.

Parameters

id an identifier of device

9.1.4.5 result_t DCCONTROL_API command_move (device_t *id*, unsigned int *pos*)

Move engine to position *pos*.

Parameters

pos position to move. Range: 0..4294967295

id an identifier of device

9.1.4.6 result_t DCCONTROL_API command_movr (device_t *id*, int *offset*)

Move engine to the distance *offset* relatively to initial position.

Parameters

offset shift from initial position. Range: -2147483647..2147483647

id an identifier of device

9.1.4.7 result_t DCCONTROL_API command_read_settings (device_t *id*)

Read all settings from controller's flash memory to controller's RAM, replacing previous data in controller's RAM.

Parameters

id an identifier of device

9.1.4.8 result_t DCCONTROL_API command_reset (device_t *id*)

Reset controller.

Can be used by manufacturer only

Parameters

id an identifier of device

9.1.4.9 result_t DCCONTROL_API command_right (device_t *id*)

Start continous moving to the right.

Parameters

id an identifier of device

9.1.4.10 result_t DCCONTROL_API command_save_settings (device_t *id*)

Save all settings from controller's RAM to controller's flash memory, replacing previous data in controller's flash memory.

Parameters

id an identifier of device

9.1.4.11 result_t DCCONTROL_API command_stop (device_t *id*)

Stop engine.

Parameters

id an identifier of device

9.1.4.12 result_t DCCONTROL_API command_update_firmware (device_t *id*, const uint8_t * *data*, uint32_t *data_size*)

Update firmware.

Service command

Parameters

id an identifier of device

data firmware byte stream

data_size size of byte stream

9.1.4.13 result_t DCCONTROL_API command_zero (device_t *id*)

Reset current position to zero.

Parameters

id an identifier of device

9.1.4.14 result_t DCCONTROL_API deinit_device (device_t *id*)

Deinit device.

Return inited device to initial state. For comunication with this device use *init_device* function ones more time

Parameters

id an identifier of device

9.1.4.15 result_t DCCONTROL_API enumerate_devices (int * *name_count*, char *** *names*, int *probe_devices*)

Enumerate all devices that looks like XIDC valid.

Value passed at *name_count* should specify size of allocated in *names* array. Strings in array *names* must be freed by *free_enumerate_devices*

Parameters

`[in,out]` ***name_count*** count of found devices

`[out]` ***names*** pointer to allocated by caller array

`[in]` ***probe_devices*** probe devices if non-zero. Range: 0..INT_MAX

9.1.4.16 result_t DCCONTROL_API free_enumerate_devices (int *name_count*, char ** *names*)

Free memory returned by *enumerate_devices*.

Value passed at *name_count* should specify size of allocated in *names* array

Parameters

`[in]` ***name_count*** count of elements in *names*

`[in]` ***names*** memory with devices info

9.1.4.17 result_t DCCONTROL_API get_analog_data (device_t *id*, analog_data_t * *ad*)

Read analog data structure that contains raw analog data from ADC embedded on board.

This function used for device testing and deep recalibraton by manufacturer only

Parameters

id an identifier of device

`[out]` ***ad*** analog data coefficients

9.1.4.18 result_t DCCONTROL_API get_calibration_coeffs (device_t *id*, calibration_settings_t * *cs*)

Read calibration coefficients.

This function fill structure with set of calibration data stored in controller's memory. There is no necessity to use it during normal operation. These data change after special device recalibraton only

See also

set_calibration_coeffs

Parameters

id an identifier of device

[out] ***cs*** calibration coefficients

9.1.4.19 result_t DCCONTROL_API get_chart_data (device_t *id*, chart_data_t * *chart_data*)

Return device electrical parameters, useful for charts.

Useful function that fill structure with snapshot of controller voltages and currents.

See also

chart_data_t

Parameters

id an identifier of device

[out] ***chart_data*** structure with snapshot of controller parameters.

9.1.4.20 result_t DCCONTROL_API get_device_information (device_t *id*, char * *manufacturer*, char * *manufacturer_id*, char * *product_description*)

Return device information.

Works with both raw or initialized device.

Parameters

id an identifier of device

[out] ***manufacturer*** manufacturer name

[out] ***manufacturer_id*** identifier of manufacturer

[out] ***product_description*** product description

9.1.4.21 result_t DCCONTROL_API get_edges_settings (device_t *id*, int * *border_flags*, int * *enders_flags*, int * *left*, int * *right*)

Read border and limit switches settings.

See also

set_edges_settings

Parameters

id an identifier of device

[out] ***border_flags*** border flags, specify types of borders and motor behaviour on borders

[out] ***enders_flags*** ender flags, specify electrical behaviour of limit switches like order and pulled positions

[out] ***left*** left border position, used if BORDER_IS_ENCODER flag is set

[out] ***right*** right border position, used if BORDER_IS_ENCODER flag is set

9.1.4.22 result_t DCCONTROL_API get_engine_settings (device_t *id*, engine_settings_t * *engine_settings*)

Read engine settings.

This function fill structure with set of useful motor settings stored in controller's memory. These settings specify motor shaft movement algorithm, list of limitations and rated characteristics

See also

set_engine_settings

Parameters

id an identifier of device

[out] ***engine_settings*** engine settings

9.1.4.23 result_t DCCONTROL_API get_feedback_settings (device_t *id*, unsigned int * *ips*, unsigned int * *feedback_flags*)

Read feedback settings.

Parameters

id an identifier of device

[out] ***ips*** number of encoder pulses per shaft revolution

[out] ***feedback_flags*** flags of feedback

9.1.4.24 result_t DCCONTROL_API get_firmware_version (device_t *id*, unsigned int * *major*, unsigned int * *minor*, unsigned int * *release*)

Read controller's firmware version.

Parameters

id an identifier of device

[out] ***major*** major version

[out] ***minor*** minor version

[out] ***release*** release version

9.1.4.25 result_t DCCONTROL_API get_home_settings (device_t *id*, home_settings_t * *home_settings*)

Read home settings.

This function fill structure with settings of calibrating position

See also

home_settings_t

Parameters

id an identifier of device

[out] ***home_settings*** calibrating position settings

9.1.4.26 result_t DCCONTROL_API get_move_settings (device_t *id*, unsigned int * *rpm*, unsigned int * *accel*, unsigned int * *tuneup_threshold*)

Read specified motor shaft speed and acceleration.

Parameters

id an identifier of device

[out] ***rpm*** motor shaft speed, RPM

[out] ***accel*** motor shaft acceleration, RPM/s

[out] ***tuneup_threshold*** threshold for enabling tuneup

9.1.4.27 result_t DCCONTROL_API get_pid_settings (device_t *id*, pid_settings_t * *pid_settings*)

Read PID settings.

This function fill structure with set of motor PID settings stored in controller's memory. These settings specify behaviour of PID routines for position, speed, voltage and current. These factors are slightly different for different positioners All boards are supplied with standart set of PID setting on controller's flash memory

See also

set_pid_settings

Parameters

id an identifier of device

[out] ***pid_settings*** pid settings

9.1.4.28 result_t DCCONTROL_API get_pwm_freq (device_t *id*, unsigned int * *freq*)

Read pwm frequency.

Parameters

id an identifier of device

[out] ***freq*** frequency

9.1.4.29 result_t DCCONTROL_API get_secure_settings (device_t *id*, unsigned int * *critical_curr*, unsigned int * *critical_voltage*, unsigned int * *critical_temp*)

Read engine voltage, current and temperature maxumum ratings.

If one of these litims exceed engine will be stopped and de-energized, corresponding flag in state_t structuse will be set

See also

state_t::flags

Parameters

id an identifier of device

[out] ***critical_curr*** maxumum current rating, if it will be exceed STATE_OVERLOAD_CURRENT flag in state_t structuse will be set

[out] ***critical_voltage*** maxumum voltage rating, if it will be exceed STATE_OVERLOAD_-VOLTAGE flag in state_t structure will be set

[out] ***critical_temp*** maxumum temperature rating, if it will be exceed STATE_OVERHEAT flag in state_t structuse will be set

9.1.4.30 result_t DCCONTROL_API get_serial_number (device_t *id*, uint32_t * *serial*)

Read device serial number.

Parameters

id an identifier of device

[out] ***serial*** serial number

9.1.4.31 result_t DCCONTROL_API get_status (device_t *id*, state_t * *state*)

Return device state.

Useful function that fill structure with snapshot of controller status, including speed, position and boolean flags

Parameters

id an identifier of device

[out] ***state*** structure with snapshot of controller status

9.1.4.32 result_t DCCONTROL_API get_sync_settings (device_t *id*, sync_settings_t * *sync_settings*)

Read synchronization settings.

This function fill structure with set synchronization settings, modes, periods and flags, that specify behaviour of synchronization In/Out. All boards are supplied with standart set of these settings

See also

set_sync_settings

Parameters

id an identifier of device

`[out]` ***sync_settings*** synchronization settings

9.1.4.33 result_t DCCONTROL_API has_firmware (device_t *id*, uint8_t * *ret*)

Check for firmware on device.

Parameters

id an identifier of device

`[out]` ***ret*** non-zero if firmware existed

9.1.4.34 result_t DCCONTROL_API init_device (device_t *id*)

Init device.

Parameters

id an identifier of device

9.1.4.35 device_t DCCONTROL_API open_device (const char * *name*)

Open a device with OS name *name* and return identifier of the device which can be used in XIDC calls.

Parameters

`[in]` ***name*** - a device name - e.g. COM3 or /dev/tty.s123

9.1.4.36 device_t DCCONTROL_API open_raw_device (const char * *name*)

Open a device with OS name *name* and return identifier of the device which can be used in XIDC calls.

Device can be raw, i.e. without firmware.

Parameters

`[in]` ***name*** - a COM device name - e.g. COM3 or /dev/tty.s123

9.1.4.37 result_t DCCONTROL_API probe_device (const char * *name*)

Check if a device with OS name *name* is XIMC device.

Be carefuly with this call because it sends some data to the device.

Parameters

[in] ***name*** - a device name

9.1.4.38 result_t DCCONTROL_API set_calibration_coeffs (device_t *id,* const calibration_settings_t * *cs*)

Set calibration coefficients.

This function send structure with set of calibration data to controller's memory. Note that all boards are supplied with full set of right calibration data. There is no necessity to change it during normal operation. These data have to be changed after special hardware device recalibraton only. If you want to recalibrate board please consult with the manufacturer. Wrong calibration settings lead to device malfunction, can lead to irreversible damage of board

See also

get_calibration_coeffs

Parameters

id an identifier of device

[in] ***cs*** calibration coefficients

9.1.4.39 result_t DCCONTROL_API set_edges_settings (device_t *id,* int *border_flags,* int *enders_flags,* int *left,* int *right*)

Set border and limit switches settings.

Parameters

id an identifier of device

[in] ***border_flags*** border flags, specify types of borders and motor behaviour on borders

[in] ***enders_flags*** ender flags, specify electrical behaviour of limit switches like order and pulled positions

[in] ***left*** left border position, used if BORDER_IS_ENCODER flag is set. Range: -2147483647..2147483647

[in] ***right*** right border position, used if BORDER_IS_ENCODER flag is set. Range: -2147483647..2147483647

9.1.4.40 result_t DCCONTROL_API set_engine_settings (device_t *id,* const engine_settings_t * *engine_settings*)

Set engine settings.

This function send structure with set of engine settings to controller's memory. These settings specify motor shaft movement algorithm, list of limitations and rated characteristics. Use it when you change motor, encoder, positioner etc. Please note that wrong engine settings lead to device malfunction, can lead to irreversible damage of board

See also

get_engine_settings

Parameters

id an identifier of device

[in] ***engine_settings*** engine settings

9.1.4.41 result_t DCCONTROL_API set_feedback_settings (device_t *id*, unsigned int *ips*, unsigned int *feedback_flags*)

Set feedback settings.

Parameters

id an identifier of device

[in] ***ips*** number of encoder pulses per shaft revolution. Range: 1..65535

[in] ***feedback_flags*** flags of feedback

9.1.4.42 result_t DCCONTROL_API set_home_settings (device_t *id*, const home_settings_t *home_settings*)

Set home settings.

This function send structure with calibrating position settings to controller's memory.

See also

home_settings_t

Parameters

id an identifier of device

[in] ***home_settings*** calibrating position settings

9.1.4.43 result_t DCCONTROL_API set_move_settings (device_t *id*, unsigned int *rpm*, unsigned int *accel*, unsigned int *tuneup_threshold*)

Set specified for this and next motion motor shaft speed and acceleration.

Parameters

id an identifier of device

[in] ***rpm*** Motor shaft speed, RPM. Range: 1..65535

[in] ***accel*** motor shaft acceleration, RPM/s. Range: 1..65535

[in] ***tuneup_threshold*** threshold for enabling tuneup. Range: 1..65535

9.1.4.44 result_t DCCONTROL_API set_pid_settings (device_t *id*, const pid_settings_t * *pid_settings*)

Set PID settings.

This function send structure with set of PID factors to controller's memory. These settings specify behaviour of PID routines for position, speed, voltage and current. These factors are slightly different for different positioners. All boards are supplied with standart set of PID setting on controller's flash memory. Please use it for loading new PID settings when you change positioner. Please note that wrong PID settings lead to device malfunction

See also

get_pid_settings

Parameters

id an identifier of device

[in] ***pid_settings*** pid settings

9.1.4.45 result_t DCCONTROL_API set_pwm_freq (device_t *id*, unsigned int *freq*)

Set pwm frequency.

Parameters

id an identifier of device

[in] ***freq*** frequency. Range: 1000..65535

9.1.4.46 result_t DCCONTROL_API set_secure_settings (device_t *id*, unsigned int *critical_curr*, unsigned int *critical_voltage*, unsigned int *critical_temp*)

Set engine voltage, current and temperature maxumum ratings.

If one of these litims exceed engine will be stopped and de-energized, corresponding flag in state_t structuse will be set. Use this function when you change motor, encoder, positioner etc. Please note that wrong maxumum ratings can lead to irreversible damage of motor

See also

state_t::flags

Parameters

id an identifier of device

[in] ***critical_curr*** maxumum current rating, if it will be exceed STATE_OVERLOAD_CURRENT flag in state_t structure will be set. Range: 1..65535

[in] ***critical_voltage*** maxumum voltage rating, if it will be exceed STATE_OVERLOAD_-VOLTAGE flag in state_t structure will be set. Range: 1..65535

[in] ***critical_temp*** maxumum temperature rating, if it will be exceed STATE_OVERHEAT flag in state_t structure will be set. Range: 0..65535

9.1.4.47 result_t DCCONTROL_API set_serial_number (device_t *id*, uint32_t *serial*, uint32_t *key*)

Write device serial number to controller's flash memory.

Can be used by manufacturer only

Parameters

id an identifier of device

[in] ***serial*** serial number. Range: 0..4294967295

[in] ***key*** protection key. Range: 0..4294967295

9.1.4.48 result_t DCCONTROL_API set_sync_settings (device_t *id*, const sync_settings_t * *sync_settings*)

Set synchronization settings.

This function send structure with set of synchronization settings, that specify behaviour of synchronization In/Out, to controller's memory. All boards are supplied with standart set of these settings

See also

get_sync_settings

Parameters

id an identifier of device

[in] ***sync_settings*** synchronization settings

9.1.4.49 result_t DCCONTROL_API write_key (device_t *id*, uint32_t *key*)

Write controller key.

Can be used by manufacturer only

Parameters

id an identifier of device

[in] ***key*** protection key. Range: 0..4294967295

Index