TOUCH documentation BASE

SPDD – Source driver

Revision 1.7 19th April 2013

www.touch-base.com\documentation\installation

Linux:- Deliverables Interfaces Libraries Integration Notes Calibration Utilities API Contact

Originally launched as OPDD (Open Pointer Device Driver) for Linux this driver has now been renamed to SPDD (Source Pointer Device Driver) for two reasons; 1: the term 'Open' can often mean 'Open source – available to all under open source licence" and our driver is not open in this sense and 2: the source code can, in theory, be made available for any OS, not just Linux.

This driver is available for environments where a source driver is required rather than the binaries installed with the standard UPDD driver. The driver could be made available for any OS but is currently only available for Linux. Should a source driver be required for a different OS please contact us to discuss your requirements.

Costs

There are potentially two sets of costs involved.

1. The cost of the source code – *which we **do not** distribute under a GNU GPL license*. Our license allows you full access to the source code for your own use as you see fit, including modifications, and caters for unlimited distribution with your systems, hardware etc. The license fee includes adding touch device support in your project. It does not allow for onward distribution of the source code.
2. Cost of any modifications required to satisfy your requirements not covered by the license fee, such as a touch interface method not supported or specialised requirements.

Please contact Sales, sales@touch-base.com, to discuss your specific requirements. The source code and deliverables will be tailored to your exact needs.

Copyright

The source code is supplied under licence to be used in systems directly associated with the purchaser of the license or its subsidiaries and customers. The source ***is not supplied under a GNU GPL license*** and therefore should not be made available to a wider audience. Each source module carries the following copyright notice that should not be removed:

```
/*
*****************************************************************************
PROGRAM ID: OPDD/SPDD
PURPOSE: Generic pointer device driver
AUTHOR: TOUCH-BASE LTD
*****************************************************************************
COPYRIGHT (c) TOUCH-BASE LTD 2010-2015. ALL RIGHTS RESERVED.
THESE NOTICES MUST NOT BE REMOVED BY SOURCE LICENSE HOLDERS.
The use or distribution of this software without express written
permission of the author is strictly prohibited.
*****************************************************************************
*/
```

This document describes the current implementation of our source offering, initially for Linux. Others will be documented as appropriate.

Linux

Our standard Linux driver installs binary modules and this is very useful for pre-configured Linux systems, non technical user, UPDD API interface, multiple controller support, simple installation etc. However, we recognise that some Linux integrations require the driver to be available in source code. To this end we have written an extensible source solution that can be modified to interface with Linux touch implementations as requested. This driver is made available, at a cost, to touch screen manufacturers and integrators on a per request basis.

Release Notes

Linux has a number of touch interfaces such as X – single and multi touch, TSlib, TUIO, evtouch, evdev and others. Then of course there are the 100's of different touch protocols implement by different touch devices. These release notes reflect the support we have implemented with each release:

Release	Date	Description
1.0.0	May 2010	Tslib interface, serial controller support
		Zytronic ZXY100 protocol definition
1.0.1	Sept 2010	X interface, generic calibration routine, USB controller support, EEprom calibration storage
		Zytronic ZXY protocol definition
1.0.2	Dec 2010	Support Data Modul USB
		API added, demo program
1.0.3	Feb 2011	EEprom read and write API
1.0.4	June 2011	ELO Smartset USB and Serial definitions added
1.0.5	April 2013	Add virtual HID (uinput) support
1.0.6	Dec 2014	Add TRS USB support

Deliverables

The full driver suite is delivered in a compressed file opdd.tgz and consists of the following files:

File	Description
opdd.ini	Settings file - can be found in the "driver" directory of the OPDD source package
opdd.c	Interface module
opdd.zip	Project
Oputil	General utilities program
Calibrate	Generic calibration program if required. i.e. TSlib has its own calibration program and utilities

ZXY100u	Firmware utility for Zytronic ZXY100 controller

Only those modules relevant to your requirements will be delivered.

The software is distributed in source form. You will need to compile the software and libraries for the target system.

Interfaces

Tslib

Tslib, available here, is an abstraction layer for touch screen panel events, as well as a filter stack for the manipulation of those events. It was created by Russell King, of arm.linux.org.uk. Examples of implemented filters include jitter smoothing and the calibration transform.

Tslib is generally used on embedded devices to provide a common user space interface to touch screen functionality. It is supported by Kdrive (aka TinyX) and OPIE as well as being used on a number of commercial Linux devices"

We have written the modules required to interface touch hardware to the Tslib abstraction layer so that any Tslib based Linux distribution or Tslib based applications will function as expected.

X

The X Window System (commonly known as X or X11) provides a windowing layer and manages the pointer device.

We have written an X interface to generate system pointer motion and mouse click emulation.

Virtual HID

Creates a Virtual touch device and passes stylus data via this device.

Library utilisation

OPDD utilises a number of software libraries:

Library	Purpose
ACE	inter process communication library
Libusb	USB device interface
QT	Development and graphics

You must install the relevant libraries on your Linux development system as outlined below.

ACE

ACE is a low level inter process communication library used by the driver and its modules. For basic processors, such as X86, we can supply the ACE library in binary format if required.

To install ACE if supplied with OPDD:

1. Open a terminal
2. Type "su" and enter the root password
3. Copy acelinux.tgz to /usr/src. eg "cp /home/user/acelinux.tgz /usr/src"
4. Type "cd /usr/src"
5. Type "tar zxvf acelinux.tgz"
6. Type "mkdir /devl/"
7. Type "ln -s /usr/src/ACE_5.6 /devl/ACE_5.6"

Note that you will need to copy the ACE library
(ACE_5.6/ACE_wrappers/ace/libACE.so.5.6.0) to a directory already included in the library search path (e.g. /usr/local/lib) or to a directory that you add to the search path yourself (e.g. the OPDD install directory).

Download from the LibACE Web page

OPDD uses ACE lib 5.6.2

This is considered an old version of the ACE library which is currently available from
http://download.dre.vanderbilt.edu/previous_versions/

Build instructions are here http://www.dre.vanderbilt.edu/~schmidt/DOC_ROOT/ACE/ACE-INSTALL.html

There are compatibility issues with later versions of ACE so it is likely that ACE libraries supplied as part of a Linux distribution are unlikely to be compatible.

One customer reported patches (with GCC 4.4.4) were needed before they were able to build ACE.

Another customer had issues building the ACE library in their environment and based on these issues we make these recommendations:

1) ACE offers two build methods, autoconf and traditional. We generally find the traditional makefile method works best.
2) If using the traditional makefile method then we suggest config-linux.h be used unless there is a more obviously relevant header for your target.
3) Some C++ compilers have trouble compiling the "dirent" functions. If compilation errors are seen that reference dirent then please use the patched file here.
4) Add the following macro to the start of config.h. This excludes an un-needed part of the ACE library that gives compile issues in some cases.
   #define ACE_HAS_POSITION_INDEPENDENT_POINTERS 0

5) If compiling on an unusual target using the traditional makefile method you might need to make changes to the ACE option macros to enable the software to build correctly.

USB lib

Utilised when handling USB devices - libusb 1.0.x required. Version 1.0.6 was used in our development.

This library may be supplied as standard as part of the Linux distribution as many recent Linux distributions have started shipping this library by default on their CD/DVD image. You may still need to select and install the library from the CD/DVD/Internet repository

The software is available at http://sourceforge.net/projects/libusb/files/libusb-1.0.

Web site is http://sourceforge.net/projects/libusb/develop

You will require gcc version 4.0.0 or later to compile this library. Extract the library source code and then run the required commands to configure, compile and install the library from the main library folder. The command sequence is typically:

```
# cd libusb-1.0.0
# ./configure
# make
# make install
# ln  -s  /usr/local/lib/libusb-1.0.so.0  /usr/lib/libusb-1.0.so.0
```

These commands may differ depending on distribution. Complete configuration information is available from the libusb links or distribution suppliers.

Users of libusb should take care to comply with the terms of the GNU LGPL as it applies to the intended usage, details are available from http://www.libusb.org.

QT library

Utilised by the calibration and general utilities program.

Version 4 required – available at http://qt.nokia.com/. Version 4.6.3 was used in our development.

This library may be supplied as standard as part of the Linux distribution. If using the Qt4 library distributed then it may be necessary to modify the file "oputils/Makefile.am" to change the search path for the Qt header files and libraries.

If it is necessary to download and build the library then building and configuration information is available from the suppliers.

Intergration

TSlib

Integrate OPDD module into Tslib

1. Copy the file "opdd.c" to the "plugins" subdirectory within your tslib source tree.
2. Modify the file "plugins/Makefile.in to add the following lines:

```
If ENABLE_OPDD_MODULE
OPDD_MODULE = opdd.la
else
OPDD_MODULE =
endif
```

Add the above lines after the similar section for "ENABLE_INPUT_MODULE"

3. Modify the line "pluginexec_LTLIBRARIES = \ ….." to add "$(OPDD_MODULE) \"

E.g.

```
pluginexec_LTLIBRARIES = \
$(LINEAR_MODULE) \
            …
            …
            $(OPDD_MODULE) \
$(INPUT_MODULE)
```

-Add the lines:-

```
opdd_la_SOURCES =          opdd.c
opdd_la_LDFLAGS =          -module $(LTSDN).
```

You should put the above lines after the line which reads:
"input_la_LDFLAGS =   -module $(LTSDN)"

4. Modify the file "configure.ac" in the root of the tslib source tree to add the following lines:-

```
AC_MSG_CHECKING([whether OPDD module is requested]) AC_ARG_ENABLE(input,
AS_HELP_STRING([--enable-opdd],
               [Enable building of OPDD module (default=yes)]),
[opdd_module=$enableval],
[opdd_module=yes])
AC_MSG_RESULT($opdd_module)
AM_CONDITIONAL(ENABLE_OPDD_MODULE, test "$opdd_module" = "yes")
```

You should add the above lines after the line:
"AM_CONDITIONAL(ENABLE_INPUT_MODULE, test "$input_module" = "yes")"

5. You can now follow the tslib instructions to build and install the library, plugins, and demo programs.
(e.g. ./autogen.sh && configure && make && make install)

If, as part of the OPDD integration, you make any changes to the ts_conf file, please be aware it is sensitive to unexpected spaces.

Configure OPDD Project

1. Uncompress the file "opdd.zip" into a directory on your system. eg /usr/src -You will need to decide where you want to install OPDD. The default location is "/opt/opdd"
2. Open the file "driver/linuxtslispointer.cpp" and find the line :

```
#define TSLIB_COM_PIPE "/opt/opdd/tslibPipe".
```

3. Modify this to reflect your installation location.

Build OPDD Project

1. Open a terminal
2. Change to the scripts directory in the source tree. eg "cd opdd/scripts"
3. Type "perl build.pl Full" to generate a full build (clean, rebuild automake files, build). or "perl build.pl Partial" to do a partial build (build). If any errors occur you can check the file "opdd/build-master-linux.log" to find out more detail (the output is also in ./<project>/.tmp.log.)
4. The "opdd" binary will be copied to the "opdd/release_linux" directory. You should copy this file to the installation directory you chose in the previous section.
5. Copy the .opdd.ini file to the installation directory.

Running the software

OPDD can be executed in any manner that is appropriate to your implementation - subject to any limitations imposed by the interface).

Typically, to run the OPDD driver type <installdir>/opdd" e.g. "/opt/opdd/opdd"

To run any of the tslib demo programs you must set the input device first.

You do this by typing the following command into the terminal:-

"export TSLIB_TSDEVICE=/opt/opdd/tslibPipe"

You can change the "/opt/opdd" section in the above command to reflect the installation path you have chosen. You can now run any of the tslib demo programs. eg "ts_print", "ts_demo", etc. To calibrate, run the demo program "ts_calibrate".

For more information about these demos see the tslib documentation.

X Interface

OPDD considerations for X

Users must have permission to be able to connect to the X.org server running on the machine otherwise the driver cannot make a connection to X and subsequently will fail to move the pointer. Further, OPDD needs to be executed by a user who has root permissions.

Configure OPDD Project

Uncompress the file "opdd.zip" into a directory on your system. eg /usr/src -You will need to decide where you want to install OPDD. The default location is "/opt/opdd"

Build OPDD Project

1. Open a terminal
2. Change to the scripts directory in the source tree. eg "cd opdd/scripts"
3. Type "perl build.pl Full" to generate a full build (clean, rebuild automake files, build). or "perl build.pl Partial" to do a partial build (build). If any errors occur you can check the file "opdd/build-master-linux.log" to find out more detail.
4. The "opdd" binary will be copied to the "opdd/release_linux" directory. You should copy this file to the installation directory you chose in the previous section.
5. Copy the .opdd.ini file to the installation directory.

Running the software

OPDD can be executed in any manner that is appropriate to your implementation - subject to any limitations imposed by the interface. The following notes are for guidance and refer to implementing on "standard" platforms. For convenience we supply a script to allow autolaunch of OPDD under X.

The file "driver/startopdd" should be copied to the install directory (e.g./opt/opdd) and made executable (chmod 755 /opt/opdd/startopdd).

The file /opt/opdd/opdd should be given root permissions ("chmod +s /opt/opdd/opdd")

If X is only being used by a single user (in the case of a media player device, etc) then:

The "$HOME/.xinitrc" of the user who will be running X should be modified to add the line "/opt/opdd/startopdd &"

If X is being used by many users and will be using a display manager then:

The file "/etc/gdm/Init/Default" should be modified to add the line "/opt/opdd/startopdd &" at the end.

User experiences

Useful user feedback or comment is documented here:

Ubuntu 10.04 integration	Library and configuration feedback

Notes

Serial Devices

You can configure the serial port that your touch screen is attached to by modifying the .opdd.ini file and changing the "port=ttyS0" line. E.g. to connect to the second com port change the line to "port=ttyS1"

Settings file

The settings file opdd.ini contains the settings used by the driver. These are documented below:

Setting	Description
[opdd]	**Driver settings**
number of devices	number of devices supported by this package, must be 1 in current implementation
scale coordinates	When true co-ordinates are scaled according to calibration data and monitor size, set to false to pass thru co-ordinates to the pointer module unchanged (e.g. when using a 3rd party pointer system)
tcpipport	TCP/IP port address for inter process communications
stabilisation	Stabilises pointer jitter. The value is expressed in pixels. E.g. with a value of 10 (hex) movements less than 16 pixels from the previous recorded point is treated as unchanged. This results in no cursor movement until a >16 difference is seen.
[opdd\1]	**Device settings**
baud	Baud rate when configured for a serial device
calxn	Calibration X data
calyn	Calibration Y data
connection type	Selects the opdd module to use to communicate with the device e.g. USB = OPDD_LINUX_USB
databits	Data bits when configured for a serial device
monitor height	Height of monitor in pixels used when scaling is in effect. The calibrate utility sets this value
monitor width	Width of monitor in pixels used when scaling is in effect. The calibrate utility sets this value
mouseport	Selects the opdd module to which output is directed – e.g. X = OPDD_LINUX_X_POINTER
parity	Parity when configured for a serial device
pid	USB product id when configured for a USB device
port	Com port name
protocol tag	Defines firmware protocol in use
stopbits	No of stop bits when configured for a serial device
vid	USB vendor id when configured for a USB device
write eeprom after cal	Indicates if calibration is automatically stored in controllers eeprom – 0 = no, 1 = yes
<eof>	End of file marker

Calibration

A number of calibration options are available under Linux and will be utilized with OPDD as required.

Interface	Calibration utility
TSlib	Use the associated ts_calibrate program.
X	We supply program "calibrate" which is used to initially calibrate the screen. A series of 4 points will be displayed which you need to touch.

Utilities

General

The general utility is called "oputils". This module implements utility functions as required.

This utility uses the QT4 library. If using the Qt4 library distributed as part of the Linux distribution then it may be necessary to modify the file "oputils/Makefile.am" to change the search path for the Qt header files and libraries.

Running oputils with no argument lists current options.

Pass the function parameter as required: oputils [parameter], e.g. optuils writeeepromcal.

Options that return data will output the value to stdio if not specified.

EEprom	This allows the storing and retrieving of EEPROM data.	
	Controller support will be added as required.	
	Sept 10	Zytronic Zypos (ZXY)USB controller supported
	Dec 10	Data Modul, EasyTouch, USB controller supported
	Feb 11	EEprom read and write functions added
Function	**Parameter**	**Description**
EEprom Storage	writeeepromcal	Store the current driver calibration data on the controller EEPROM for retrieval at a later date.
EEprom Retrieval	readeepromcal	Read stored calibration data from the controller EEPROM and use it as the current calibration settings.

Notes:

1. The storage of EEPROM calibration data can be automated by modifying the .opdd.ini file and changing the "write eeprom after cal" setting to 0x00000001.

EEprom Read	readeeprom <val>	Reads data from eeprom
	(device,addr.length)	Example: 'Oputils readeeprom 1 0 4' returns 4 bytes of data starting at address 0 on device 1
EEprom Write	writeeeprom <val>	Writes data to eeprom
	(device,Addr,data)	Example: 'opuutils writeeeprom 1 0 FF FE FD FC FB FA writes the 6 bytes 0xff – 0xfa to address 0 – 5 on device 1

Notes:

1. The underlying read / write code must be relevant to the controller in use as firmware eeprom read / write commands will be specific to each controller type.
2. The source code of oputils serves as a source example should you need to read or write to eeprom from your own code.

Specialised

We also can supply / develop source utilities as required. This section lists available specialised utilities:

Zytronic ZXY100 serial firmware setting updates

Utility used to update firmware settings in the Zytronic ZXY100 serial controller. This is supplied when using the ZXY100 serial controller.

The list of supported firmware commands is:-

zxy100u		The sensitivity of the touch screen is controlled using the Sensitivity and Threshold parameters below:
	{sensitivity [n]}	The Coarse sensitivity can be changed between Option 1 to 4. This should be initially used to set the overall coarse sensitivity of the sensor with the different options relating to the sensor glass/overlay glass thickness used (guideline glass thickness is displayed in millimetres along side each Option 1 to 4). Increasing the Coarse Sensitivity will make the touch sensor more sensitive to an applied touch.
	{threshold [n]}	The Threshold control can then be used to set the fine sensitivity of the sensor. The Threshold can be changed between a value of 12 and 100. Increasing the Threshold will make the sensor less sensitive to an applied touch and decreasing the threshold value will make the sensor more sensitive to an applied touch. **Please note:** the Threshold control has the inverse affect to that of the Coarse sensitivity control.
	{equalise }	This forces the touch screen controller to normalise all the wire levels within the sensor and take these normalised levels as new reference values for processing information.
	{factoryreset }	This will initiate a reset of the touch screen controller. After execution the controller may take a few seconds to re-establish full operation to an applied touch.
	{restore }	Restore factory default settings
	{version }	Return firmware version number
	{update xxx.zyf }	Update controller's firmware from firmware update file

options that take an optional numeric value [n] will:

#set the value if specified
#output the current value to stdio if not specified

Application Programming Interface

OPDD supports a programming interface to allow client programs to communicate with the driver.
The API is implemented by the Client class and communicates with the driver using a TCP/IP link bound to localhost (127.0.0.1).

The driver supports a number of commands which are implemented in ./drier/command.cpp.
These are mainly for internal use and not documented. A programmer can use this (and the corresponding client side code in the supplied source) to extent the API as needed.

The supported commands are encapsulated in the public methods of the Client class:

```
public Client::Client();

      The constructor for the client class.
      Takes no arguments.

public Client::~Client();
      The destructor for the client class.
      Any open session is closed and stream mode terminated (output is directed to the system
      mouse interface).

public int Client::open();

      Opens the TCP/IP link to the driver.
      Defaults to port 4142, but this can be altered in the.opdd.ini file if one exists in the
      working directory.
      Returns 0 for success, -1 for failure.

public bool Client::StartStreamTouch();
      Initiates stream touch mode. All touch data is directed to this client instead of the system
      mouse interface.
      Returns false in the event of an error.

public bool Client:: ReadStreamTouch(int& x,int& y,int& stylus,bool& touching
      Reads data from the touch stream (initiated by StartStreamTouch())
      Arguments
            x – returns the x co-ordinate for the current touch location
            y – returns the y co-ordinate for the current touch location
            stylus – for a multi touch controller returns the stylus number (0 for non multi-
            touch)
```

```
            touch – returns true if contact is currently taking place
      Returns true if data is returned.
      This function is expected to be called in worker thread and returns false ever second or so
      to allow an opportunity for thread termination.

public bool Client::WriteEEPROM(int aDevice, int address, int length, const unsigned char* data);

      If the device identified by aDevice supports eeprom writes then the data of specific length
      is written to the address specified.
            The meaning of “address” and the valid values will be dictated by the controller
            implementation.

      NB opdd currently only supports a single device, so aDevice must be 1.

public bool Client::ReadEEPROMProlog(int aDevice, int address, int length);

      If the device identified by aDevice supports eeprom reads then a read operation for data of
      specific length at the address specified is initiated. The caller should wait for the
      completion of this operation using the “eepromcalstate” command, then use ReadEEPROMEpilog
      to complete the read.
            The meaning of “address” and the valid values will be dictated by the controller
            implementation.

      See oputils source code for a detailed example of the use of this api.

      NB opdd currently only supports a single device, so aDevice must be 1.

public bool Client::ReadEEPROMEpilog(unsigned char* data);

      Complete a read operation initiated by Client::ReadEEPROMProlog(). The data  is copied to
      the addressed by the data argument. This must be a caller provided block of memory of (at
      least) the size specified by the length argument to Client::ReadEEPROMProlog()
```

Example code

The following code reads touch data in a worker thread.
This is taken from the Scribble example described below.

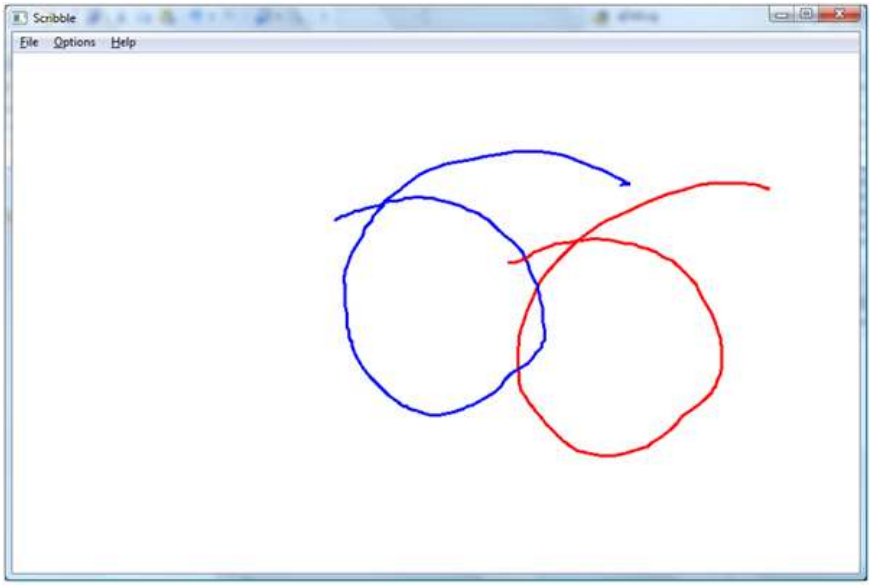
```
ACE_THR_FUNC_RETURN ReadThread(void* aArg)
{
  Client client;              // construct a client interface object
  if(client.open() == -1)     //open a connection with the driver
  {
    return(0);                // something when wrong, bale out
  }
  if(!client.StartStreamTouch())    // start touch mode
  {
    return(0);                // something when wrong, bale out
  }

  while(theRunReadThread)
  {
    TouchEvent* t = new TouchEvent;
    if(client.ReadStreamTouch(t->x,t->y,t->stylus,t->touching))   // read the touch data
    {
      QApplication::postEvent( (QWidget*)aArg, t);                // do something with the data read
    }
    else
    { // in the idle state ReadStreamTouch() will return false every second or so to allow thread
termination to take place

    }
  }
  return(0);
}
```

Example program

An example program is available to illustrate the use of the touch stream mode.
Scribble, located in ./examples/scribble, is an adaptation of the Qt scribble example.
Full instructions to build Qt examples are available in the Qt documentation, but generally you will use the following commands from the scribble directory

```
qmake
make (or nmake on Windows)
```

Run the application to see the application shown below. This is a multi touch aware drawing application.
The current example supports a maximum of 2 touches, but the API itself is unlimited.



Contact

For further information or technical assistance please email the technical support team at technical@touch-base.com