

**Lexmark Linux Printer Driver Developer's Kit  
X5250/X7170 Sample SANE Scanner Driver**

**Version 2.0  
January 31, 2005**

**Lexmark International  
Lexington, KY 40550**

---

## **Preface**

---

### **Document Owner**

---

### **Document Editor**

---

### **Contributors**

---

### **Document Changes**

July 07, 2004	Version 1.0	Created
January 31, 2005	Version 2.0	Modified for x7170

---

### **References**

---

### **Notes**

## Table of Contents

<b>PREFACE .....</b>	<b>2</b>
<b>TABLE OF CONTENTS .....</b>	<b>3</b>
<b>CHAPTER 1 REFERENCE .....</b>	<b>5</b>
<b>CHAPTER 2 FILES AND DIRECTORIES.....</b>	<b>6</b>
<b>CHAPTER 3 TOOLS.....</b>	<b>7</b>
<b>CHAPTER 4 BUILDING THE SAMPLE SCANNER DRIVER.....</b>	<b>8</b>
<b>CHAPTER 5 X5250/X7170 SAMPLE DRIVER INTRODUCTION.....</b>	<b>9</b>
<b>CHAPTER 6 GLOBAL ARCHITECTURE .....</b>	<b>10</b>
6.1 SANE OVERVIEW.....	10
6.1.1 SANE Frontend.....	10
6.1.2 SANE Backend.....	10
6.1.3 Configuration Files.....	10
6.1.4 SANE API.....	11
6.2 SCANNING PROCESS.....	11
6.2.1 Nominal Case.....	11
6.2.2 Error case .....	11
<b>CHAPTER 7 FUNCTIONAL DESCRIPTION .....</b>	<b>12</b>
7.1 OVERVIEW.....	12
7.2 LEXMARK SANE BACKEND .....	12
7.3 LLPDDK WRAPPER.....	13
7.4 COMMUNICATIONS .....	13
7.5 ERROR MANAGER.....	13
7.5.1 Error Codes .....	13
7.6 LEXMARK LINUX PRINTER DRIVER DEVELOPER'S KIT.....	13
<b>CHAPTER 8 DETAILED IMPLEMENTATION.....</b>	<b>14</b>
8.1 SUB COMPONENT INTERACTION .....	14
8.2 LEXMARK SCANNER DRIVER .....	14
8.2.1 Lexmark Backend.....	14
8.2.2 LLPDDK Wrapper.....	14
8.3 SANEPORTMONITOR .....	17
8.3.1 Behavior.....	18
8.4 SCANERRORCOMMUNICATOR .....	18
8.4.1 Behavior.....	18
8.5 LEXMARK LINUX PRINTER DRIVER DEVELOPER'S KIT.....	19
8.5.1 Behavior.....	19
<b>CHAPTER 9 APPLICATION PROGRAMMING INTERFACE .....</b>	<b>20</b>
9.1 LEXMARK SANE BACKEND (X5250/X7170 BACKEND) .....	20
9.1.1 sane_init ().....	20
9.1.2 sane_exit ().....	20
9.1.3 sane_get_devices ().....	21
9.1.4 sane_open ().....	21
9.1.5 sane_close () .....	22
9.1.6 sane_get_option_descriptor ().....	22
9.1.7 sane_control_option ().....	23
9.1.8 sane_get_parameters ().....	23

---

9.1.9 <i>sane_start()</i> .....	24
9.1.10 <i>sane_read ()</i> .....	25
9.1.11 <i>sane_cancel()</i> .....	25
9.1.12 <i>sane_set_io_mode()</i> .....	26
9.1.13 <i>sane_get_select_fd()</i> .....	26
9.1.14 <i>init_options()</i> .....	27
9.1.15 <i>attachLexmark()</i> .....	27
9.2 SANEPORTRMONITOR .....	28
9.2.1 <i>SanePortMonitor::SanePortMonitor()</i> .....	28
9.2.2 <i>SanePortMonitor::~~SanePortMonitor()</i> .....	28
9.2.3 <i>SanePortMonitor::PM_Open()</i> .....	29
9.2.4 <i>SanePortMonitor::PM_Close()</i> .....	29
9.2.5 <i>SanePortMonitor::PM_Write()</i> .....	29
9.2.6 <i>SanePortMonitor::PM_Read()</i> .....	30
9.2.7 <i>SanePortMonitor::PM_SetDeviceName()</i> .....	30
9.2.8 <i>SanePortMonitor::PM_SetDeviceName()</i> .....	30
9.2.9 <i>SanePortMonitor::PM_AttachDevice()</i> .....	31
9.3 SCANERRORCOMMUNICATOR .....	31
9.3.1 <i>ScanErrorCommunicator ::ScanErrorCommunicator()</i> .....	31
9.3.2 <i>ScanErrorCommunicator ::~ScanErrorCommunicator()</i> .....	32
9.3.3 <i>ScanErrorCommunicator ::ReportError()</i> .....	32
9.3.4 <i>ScanErrorCommunicator ::GetUserDecision()</i> .....	32
9.4 LEXMARK LINUX PRINTER DRIVER DEVELOPER'S KIT .....	33

---

## Chapter 1 REFERENCE

In this document, we make reference to the following documentation:

- LDK Interface Spec (v. 2.0-4)
- SANE Documents (<http://www.sane-project.org/docs.html>)

## Chapter 2 FILES AND DIRECTORIES

A number of files and directories represent the architecture of the X5250/X7170 Sample SANE Scanner Driver/s based on the Lexmark Linux Printer Driver Developers Kit. The following lists the most important files and directories included in the package and their locations:

[install\_dir]/README

- ❑ This file informs the user about the SANE Scanner Driver. It also includes information such as requirements, basic installation, prerequisites and other information necessary to use the driver.

[install\_dir]/COPYING

- ❑ The file containing the text version of the GNU General Public License (GPL), which governs the copying agreement of the Sample SANE Scanner Driver based on the Lexmark Linux Printer Driver Developer's Kit.

[install\_dir]/backend

- ❑ This directory contains the configuration file and source code which implements the SANE APIs required to support a scanner driver in Linux.

[install\_dir]/lexmark

- ❑ This directory contains additional source code other than those required by SANE. In particular, these files are used to interface to the LLPDDK library.

[install\_dir]/daemon

- ❑ This directory contains the source code for the scan button daemon.

[install\_dir]/include

- ❑ This directory contains SANE header files that are part of the SANE package and are used in the backend driver as include files.

[install\_dir]/sanei

- ❑ This directory contains SANE source files that are part of the SANE package and are included in the driver for compilation purpose.

[install\_dir]/setapps

- ❑ This directory contains the source code for setting the scan applications in the OP panel of AIO's like X7170.

[install\_dir]/adfutil

- ❑ This directory contains the source code which implements the ADF support of AIO's like X7170.

The “[install\_dir]” refers to the directory where the X5250/X7170 sample driver package was expanded.

---

## Chapter 3 TOOLS

The X5250/X7170 sample scanner driver was developed with the following tools:

### Computer System

- ▶ Dell 4100 Computers
- ▶ Linux Operating System
- ▶ Red Hat Linux Distribution 8.0
- ▶ Kernel 2.4.18-14

### Computer Applications

- ▶ Text Editors (vi, vim, emacs, etc)
- ▶ KDevelop, Anjuta

---

## Chapter 4 BUILDING THE SAMPLE SCANNER DRIVER

Prior to compiling the sample scanner driver, the user will have to install first the USB scanner device driver. After downloading LEXUSB-SCANNER-x.y-z.tar.gz, type at the command prompt the following:

- ▶ `tar xvfz LEXUSB-SCANNER-x.y-z.tar.gz`
- ▶ `cd LEXUSB-SCANNER-x.y-z`
- ▶ `make` (Please refer to the README file included in the tarball for more details)
- ▶ `make install`

To compile and build the sample scanner driver package, type at the command prompt the following:

- ▶ `tar xvfz X---SampleSANE-x.y-z.tar.gz`
- ▶ `cd X---SampleSANE-x.y-z`
- ▶ `./configure`
- ▶ `make`
- ▶ `make install`

Please refer to the README file included for a detailed explanation. Please take note that the X5250/X7170 LLPDDK binaries should be installed in the system prior to compiling the sample driver.

---

## Chapter 5 X5250/X7170 SAMPLE DRIVER INTRODUCTION

The purpose of this document is to define the architecture for the X5250/X7170 Sample SANE Scanner Driver based on the Lexmark Linux Printer Driver Developer's Kit (LLPDDK) binaries.

The document is divided in five (4) parts:

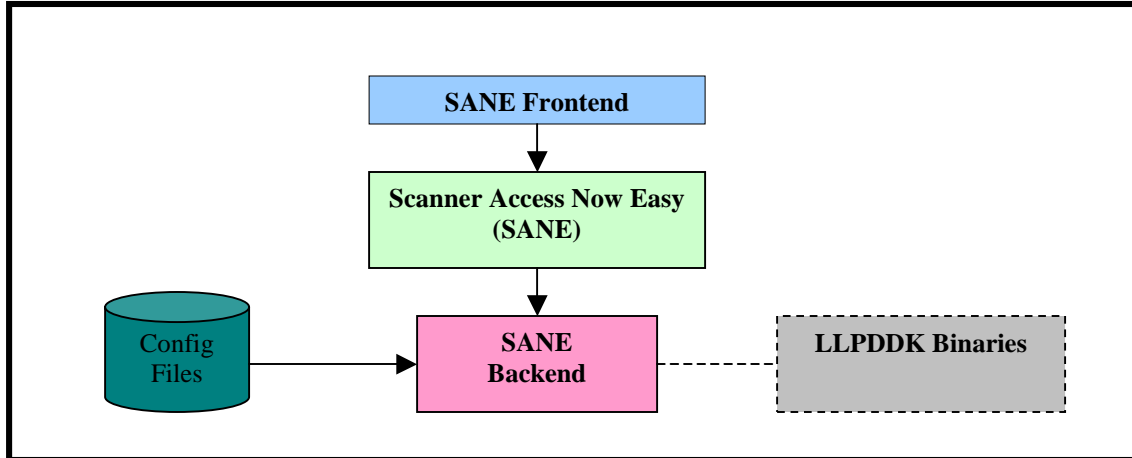
First, the global architecture (see [Chapter 6](#)) exposed: we present the role of the different scanner components.

The functional description of the sample driver components (see [Chapter 7](#) and [Chapter 8](#)) discussed in greater details.

Lastly, the Application Programming Interface (API) of each component (see [Chapter 9](#)) is provided as reference to the user of the sample SANE scanner driver.

## Chapter 6 GLOBAL ARCHITECTURE

In this part, the global architecture of the X5250/X7170 Sample Scanner Driver (SANE) using the LLPDDK is described.



**Figure 6-1 SANE Block Diagram**

SANE frontends are applications developed by third-party vendors that are readily available for use by backend developers such as Lexmark.

The X5250/X7170 scanner driver makes use of the SANE APIs to implement the scanning capability in Linux. Scanner Access Now Easy (SANE) is an application-programming interface that provides standardized access to any raster image scanner hardware. These SANE APIs are implemented by the X5250/X7170 SANE backend and are called by the SANE frontend used.

The X5250/X7170 SANE backend is also responsible for encapsulating the Lexmark scanner device and its functionalities. It makes use of the X5250/X7170 LLPDDK binaries to communicate with the device.

## 6.1 SANE OVERVIEW

### 6.1.1 SANE Frontend

The SANE frontend is an application that uses the SANE interface. Many frontends that are developed by third-party developers are readily available for use by our backend. Such frontends include Xsane, Xscanimage, and QuiteInsane.

### 6.1.2 SANE Backend

The SANE backend is a driver that implements the SANE interface. This is sometimes referred to as our scanner driver. The X5250/X7170 SANE invokes the X5250/X7170 LLPDDK binaries to communicate with our scanner device.

### 6.1.3 Configuration Files

The configuration files consist of:

- dll.conf
- saned.conf

dll.conf lists all the backend supported by SANE. The user must edit this file and add “lexmark” in the list of supported device to enable the use of our scanner driver.

saned.conf file is a list of hostnames or IP addresses that are permitted by saned to use local SANE devices in a networked configuration.

## 6.1.4 SANE API

The SANE API provides standardized access to any raster image scanner hardware. This standardized interface allows developers to write just one driver for each scanner device instead of one driver for each scanner and application.

---

## 6.2 SCANNING PROCESS

### 6.2.1 Nominal Case

During a nominal scanning, the following operations are performed:

- The scanner is locked prior to scanning.
- The backend sends a command to the scanner to define the color format of the pixels.
- The backend sends a command to the scanner to define the scanning resolution.
- The backend sends a command to the scanner to define the offset of the top-left corner of the rectangle to be scanned from the top-left of the scanner bed.
- The backend sends a command to the scanner to define the size of the rectangle to be scanned.
- The backend sends a command to the scanner to set the image filtering required by the host for the next scan job.
- The backend sends a command to the scanner to set the clipping, brightness and contrast levels for the next scan job.
- The backend sends a command to the scanner to initiate the scan.
- The backend reads the scan data returned by the scanner until there is no more data.
- The backend sends a command to the scanner to end the scan after the rest of the scan data has already been read.
- The scanner is unlocked to prepare the scanner for next job.

### 6.2.2 Error case

In case of error in any of the steps prior to initiating a scan job, the backend reports the error ID to SANE and the scan job is aborted. These errors are defined by SANE and are reported by the frontend using its own error strings. The backend reports these errors in a more descriptive manner in its own separate log file, /var/log/scanerror\_log.

In case when the user cancelled the scan job in the frontend while the scanner has already been started and scan data is already being sent, the backend will send a command to the scanner to abort the scan job and the application is forced-quit. When canceling of scan job is invoked in the device, the backend will send an error ID to SANE and scan job is aborted. The frontend will display the error encountered but the application is not forced-quit.

## Chapter 7 Functional Description

This chapter deals with the functional aspect of the Linux Sample Scanner Driver. It defines the modules that will be explained later in this document.

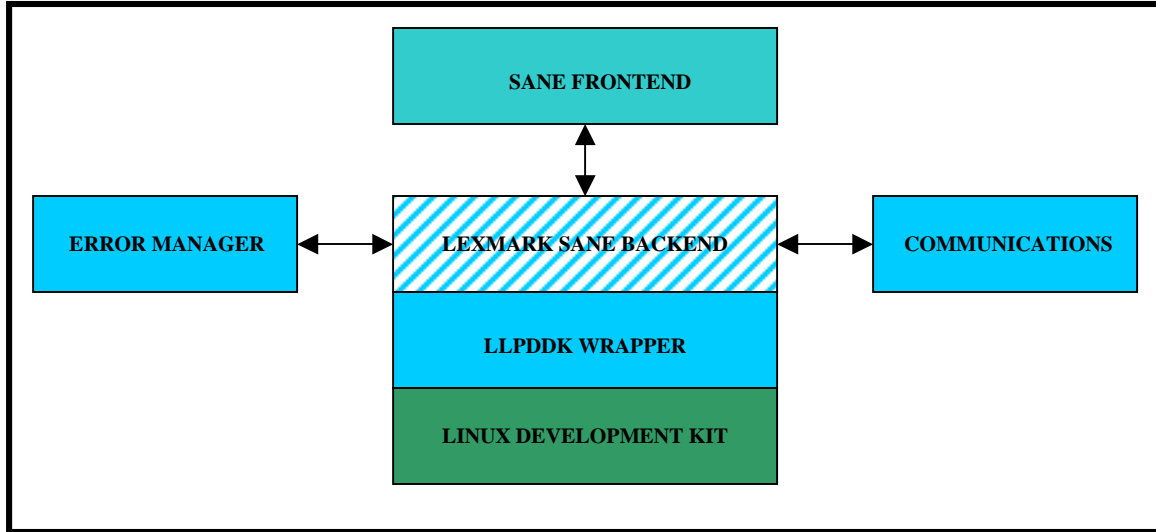


Figure 7-1 SANE Sample Driver Functional Diagram

### 7.1 Overview

The architecture of the X5250/X7170 Linux Sample SANE Scanner Driver is based on four software components:

1. **Lexmark SANE Backend** implements the SANE APIs that are required by SANE for backend developers to support our scanner. It is implemented purely in C and defines options that describe the capabilities of our scanner driver.
2. The **LLPDDK Wrapper** is a C++ wrapper responsible for interfacing to the LLPDDK binaries.
3. The **Error Manager module** implements the error reporting developed from the LLPDDK. It is derived from the virtual class defined in the platform independent driver. It is used to notify errors occurring in the current process to the outside world.
4. The **Communications** is in charge of establishing the communication path between the X5250/X7170 scanner backend and the printer device. It is also used to send or read raw data to/from the printer device.

### 7.2 Lexmark SANE Backend

The Lexmark SANE Backend implements all the required callback SANE APIs needed to support scanning in Linux for Lexmark scanners. It integrates with the X5250/X7170 LLPDDK Binaries through the LLPDDK Wrapper since the binaries are implemented in C++. The Backend also defines device-specific options or properties that our device supports. It also receives the settings for the next scan job from the frontend as SANE options and passes these settings to the LLPDDK.

## 7.3 LLPDDK Wrapper

The LLPDDK wrapper is a C++ wrapper to the LLPDDK binaries. It maps to LLPDDK the properties set from the frontend and passed by the Lexmark SANE backend as SANE options to this module.

---

## 7.4 Communications

This module is in charge of communicating with the scanner over USB. It is derived from the virtual class defined in the platform independent driver, called the PortMonitor, and thus has the same interface. The PortMonitor is created by the LLPDDK Wrapper module and is used as communication interface with the scanner by the LLPDDK.

---

## 7.5 Error Manager

The Error Manager provides the mechanism of notifying the outside world that an error was encountered. It is derived from the virtual class defined in the platform independent driver called ScanErrorInterface. Instead of displaying the error message in the frontend's GUI, error messages specific to the device is logged in /var/log/scanerror\_log.

---

### 7.5.1 Error Codes

The following error codes are reported to a log file (/var/log/scanerror\_log):

- ERR\_NONE - scanner encountered no errors
- ERR\_COMMUNICATION\_ERROR - scanner gets communication errors  
This may be a result of an offline condition  
Or the connection to the scanner is broken
- ERR\_DEVICE\_WARMING\_UP - scanner device is warming up
- ERR\_SCANNER\_NOT\_LOCKED - device failed to lock for scanning
- ERR\_COPIER\_NOT\_LOCKED - device failed to lock for copying
- ERR\_SCAN\_STOPPED - scan stopped from device
- ERR\_DEVICE\_OFF - scanner currently powered off
- ERR\_MEMORY\_OUT - scanner is out of memory
- ERR\_ADF\_PAPER\_JAM - paper jam has occurred in ADF (for X7170)
- ERR\_UNEXPECTED\_ERROR - unexpected error occurred

---

## 7.6 Lexmark Linux Printer Driver Developer's Kit

This component represents the internal workings of a Lexmark specific scanner device. This information is proprietary and thus, was packaged as binaries. This driver developer's kit is target to be used in an X86 based computer system. Please see LLPDDK Interface Spec (v. 2.0-4) for more details.

---

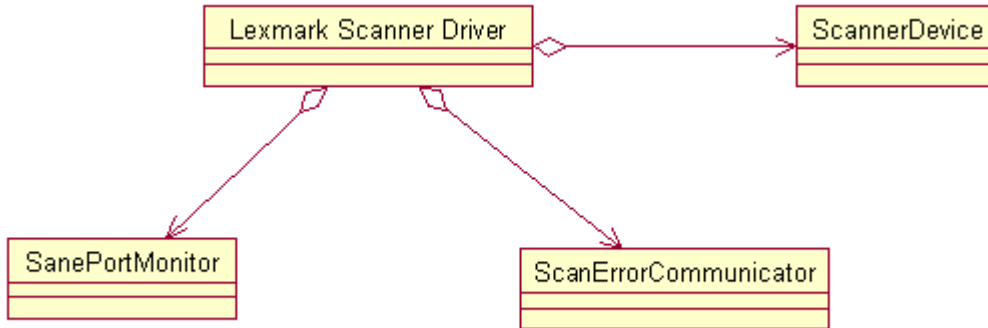
## Chapter 8 DETAILED IMPLEMENTATION

This part intends to give a detailed description of each module and their relationship with one another. The exact API of each module is also described.

---

### 8.1 SUB COMPONENT INTERACTION

The following UML plot underlines the relationships between the different components of the implementation specific to Linux.




---

### 8.2 Lexmark Scanner Driver

The Lexmark Scanner Driver is mainly composed of the Lexmark backend driver that is required by SANE to be implemented by all scanner developers and the LLPDDK wrapper. The scanner driver is called by the SANE frontend in use. The scanner driver is responsible for setting all properties for the scan job and mapping those properties to the ScannerDevice's (LLPDDK) scan settings. It also instantiates the abstract classes ScanErrorInterface and PortMonitor to make the communication and error interface specific to scanning.

#### 8.2.1 Lexmark Backend

The purpose of the Lexmark backend is to implement the required SANE APIs that will be called by the particular frontend in use. It is up to the SANE frontend how and when these APIs will be called as long as it follows SANE code flow for calling the APIs. The backend has no error checking to determine if the APIs were called according to code flow for the SANE APIs.

#### 8.2.2 LLPDDK Wrapper

The LLPDDK wrapper is part of the Lexmark Scanner Driver as a separate component who is mainly responsible for interfacing to the LLPDDK and instantiating the error manager and communications objects. This is done, as a separate component apart from the Lexmark Backend since it is required that backend drivers must be implemented purely in C. It also maps the SANE options set in the backend to the LLPDDK specific settings for the scan job.

##### 8.2.2.1 Behavior

###### 8.2.2.1.1 Initialization

Component initialization should take place prior to any operations in scanning. This is done when SANE calls the initialization API (`sane_init`) that is called at the beginning of every scan job. At this point,

initialization of the LLPDDK should take place. During initialization, the SanePortMonitor and ScanErrorCommunicator are also instantiated after which the ScannerDevice is instantiated after successful creation of these objects. Aside from the two objects, ScannerDevice also requires the scanner model for successful interface to the LLPDDK.

```

...
...
SANE_Status
sane_init (SANE_Int * version_code, SANE_Auth_Callback authorize)
{
...
...
    status = llpddk_init ();
...
}

extern "C" SANE_Status llpddk_init()
{
    sanepm = new SanePortMonitor;
    if (sanepm == NULL)
        return SANE_STATUS_INVALID;

    ei = new ScanErrorCommunicator;

    if (ei == NULL)
    {
        delete sanepm;
        return SANE_STATUS_INVALID;
    }

    sd = new ScannerDevice(sanepm, ei, "X5250");

    if (sd == NULL)
    {
        delete sanepm;
        delete ei;
        return SANE_STATUS_INVALID;
    }

    return SANE_STATUS_GOOD;
}
...
...

```

### 8.2.2.1.2 Setting the Job Properties

Another important task of the Lexmark Scanner Driver is to set the scan job properties prior to initiating a scan. This information will be helpful in producing the desired scan output by sending these scan properties as command understandable by the scanner firmware. This is done in the LLPDDK wrapper (llpddk\_start\_scan) prior to sending the command to initiate the scan to the device. Properties that are set include the bits per channel, channel, halftone, horizontal and vertical dpi, and scan rectangle coordinates.

```

...
...
extern "C" SANE_Status llpddk_start_scan(
    Option_Value opt[])
{
    ScannerDevice::SD_ErrCode sderr;
    ScannerDevice::SD_ScanProp sdprop;

    // bits per channel
    sdprop.bitsperchannel = ScannerDevice::SD_EIGHT_BPC;

    // channels
    sdprop.channels = ScannerDevice::SD_RGB_CHANNEL;
}

```

```

if (strcmp(opt[LXK_OPT_CHANNELS].s,"Gray") == 0)
    sdprop.channels = ScannerDevice::SD_MONO_CHANNEL;

// halftone
sdprop.halftonemode = ScannerDevice::SD_ERROR_DIFFUSION;
if (strcmp(opt[LXK_OPT_HALFTONE_MODE].s,"Threshold") == 0)
    sdprop.halftonemode = ScannerDevice::SD_THRESHOLD;

// compression
sdprop.compressionmode = ScannerDevice::SD_UNCOMPRESSED;

// horizontal dpi
unsigned int x_dpi, y_dpi;
switch (opt[LXK_OPT_X_DPI].w)
{
    case 75: sdprop.horizontaldpi = ScannerDevice::SD_75_DPI;
            x_dpi = 75;
            break;
    case 100: sdprop.horizontaldpi = ScannerDevice::SD_100_DPI;
            x_dpi = 100;
            break;
    case 150: sdprop.horizontaldpi = ScannerDevice::SD_150_DPI;
            x_dpi = 150;
            break;
    case 300: sdprop.horizontaldpi = ScannerDevice::SD_300_DPI;
            x_dpi = 300;
            break;
    case 600: sdprop.horizontaldpi = ScannerDevice::SD_600_DPI;
            x_dpi = 600;
            break;
    case 1200: sdprop.horizontaldpi = ScannerDevice::SD_1200_DPI;
            x_dpi = 1200;
            break;
    case 2400: sdprop.horizontaldpi = ScannerDevice::SD_2400_DPI;
            x_dpi = 2400;
            break;
    case 4800: sdprop.horizontaldpi = ScannerDevice::SD_4800_DPI;
            x_dpi = 4800;
            break;
    default: sdprop.horizontaldpi = ScannerDevice::SD_150_DPI;
            break;
}

// vertical dpi
switch (opt[LXK_OPT_Y_DPI].w)
{
    case 75: sdprop.verticaldpi = ScannerDevice::SD_75_DPI;
            y_dpi = 75;
            break;
    case 100: sdprop.verticaldpi = ScannerDevice::SD_100_DPI;
            y_dpi = 100;
            break;
    case 150: sdprop.verticaldpi = ScannerDevice::SD_150_DPI;
            y_dpi = 150;
            break;
    case 300: sdprop.verticaldpi = ScannerDevice::SD_300_DPI;
            y_dpi = 300;
            break;
    case 600: sdprop.verticaldpi = ScannerDevice::SD_600_DPI;
            y_dpi = 600;
            break;
    case 1200: sdprop.verticaldpi = ScannerDevice::SD_1200_DPI;
            y_dpi = 1200;
            break;
    case 2400: sdprop.verticaldpi = ScannerDevice::SD_2400_DPI;
            y_dpi = 2400;
            break;
    case 4800: sdprop.verticaldpi = ScannerDevice::SD_4800_DPI;
            y_dpi = 4800;
            break;
    default: sdprop.verticaldpi = ScannerDevice::SD_150_DPI;
}

```

```

        break;
    }
    double tl_x, tl_y, width_in, height_in;

    //in inches
    tl_x = (double) (opt[LXK_OPT_TL_X].w / MM_PER_INCH);
    tl_y = (double) (opt[LXK_OPT_TL_Y].w / MM_PER_INCH);
    width_in = (double) (opt[LXK_OPT_BR_X].w / MM_PER_INCH) - tl_x;
    height_in = (double) (opt[LXK_OPT_BR_Y].w / MM_PER_INCH) - tl_y;

    // offset in px
    sdprop.horizontaloffset = (unsigned int) (tl_x * x_dpi);
    sdprop.verticaloffset = (unsigned int) (tl_y * y_dpi);

    // size in px
    sdprop.width = (unsigned int)(width_in * x_dpi);
    sdprop.height = (unsigned int)(height_in * y_dpi);
-
-
}
...
...

```

### 8.2.2.1.3 Initiating the Reading of the Scan Data

Once the scan properties have already been sent, the Lexmark Scanner Driver is ready to initiate the scan and receive and process the scan data. Once the command to start the scan has already been sent, the driver is now ready to invoke the reading of data.

```

...
...
extern "C" SANE_Status llpddk_start_scan(
    Option_Value opt[])
{
-
-
    sderr = sd->SD_StartScan(sdprop);
    if (sderr != ScannerDevice::SD_OK)
        return SANE_STATUS_INVAL;

    return SANE_STATUS_GOOD;
}

extern "C" long llpddk_read_scan_data(
    SANE_Byte * data,
    SANE_Int size,
    int * eof)
{
    return sd->SD_ReadScanData(
        (unsigned char *) data,
        (unsigned long) size,
        eof);
}
...
...

```

## 8.3 SanePortMonitor

The SanePortMonitor is the communication component of the sample SANE scanner driver that provides bi-directional communication between the host computer and the Lexmark scanner device. It has the capability of writing and reading information to and from the scanner device. Besides writing and reading data, the communications module knows how to open and close the scanner port.

This component is a user-specific implementation of the abstract module PortMonitor of the LLPDDK. For the sample SANE scanner device, we termed it as the SanePortMonitor to refer to its implementation of communication using SANE. Please refer to the LLPDDK interface document for more information of the PortMonitor module.

The LinuxPortMonitor retrieves a string with a location name. This location name is expected to be the path and name of virtual device of the Linux file system representing a scanner connected to a USB port.

### 8.3.1 Behavior

#### 8.3.1.1 Initialization

The SanePortMonitor is created by the LLPDDK Wrapper during initialization. No scanner port is specified yet, it just initialize the sanei functions that will be used for communicating with the scanner. If initialization of this module fails, the driver returns immediately and terminates the scan.

```

...
extern "C" SANE_Status llpddk_init()
{
    sanepm = new SanePortMonitor;
    if (sanepm == NULL)
        return SANE_STATUS_INVALID;
}

SanePortMonitor::SanePortMonitor()
{
    // make sure port handle is in default state
    m_PortHandle = SANE_PM_INVALID_HANDLE;

    // as well as the device name
    m_DeviceName = NULL;

    // initialize sanei functions
    sanei_usb_init();
}
...

```

## 8.4 ScanErrorCommunicator

The ScanErrorCommunicator provides error management of the sample SANE scanner driver.

Like the SanePortMonitor, the ScanErrorCommunicator is a user-specific implementation of the abstract module ScanErrorInterface of the LLPDDK. Please refer to the LLPDDK interface document for more information of the ScanErrorInterface.

### 8.4.1 Behavior

#### 8.4.1.1 Initialization

The ScanErrorCommunicator is created by the LLPDDK Wrapper and initialized by the LLPDDK. After successful creation, the LLPDDK will readily use this module for error management.

```

-
extern "C" SANE_Status llpddk_init()
{
-

```

```

-
    ei = new ScanErrorCommunicator;

    if (ei == NULL)
    {
        delete sanepm;
        return SANE_STATUS_INVAL;
    }
-
-
}
-
-

```

---

## 8.5 Lexmark Linux Printer Driver Developer's Kit

The ScannerDevice component (of the LLPDDK) is the interface of the Lexmark Scanner Driver to the Lexmark Linux Printer Driver Developer's Kit. It provides attachment of important components such as the SanePortMonitor and ScanErrorCommunicator. These Linux-specific implementations are important for proper functioning of the LLPDDK.

### 8.5.1 Behavior

#### 8.5.1.1 Initialization

The LLPDDK Wrapper initializes the ScannerDevice. In order to initialize it properly, the component should be able to initialize properly first the SanePortMonitor and ScanErrorCommunicator components because all these components should be passed as parameters to the constructor of ScannerDevice.

```

extern "C" SANE_Status llpddk_init()
{
    sanepm = new SanePortMonitor;
    if (sanepm == NULL)
        return SANE_STATUS_INVAL;

    ei = new ScanErrorCommunicator;

    if (ei == NULL)
    {
        delete sanepm;
        return SANE_STATUS_INVAL;
    }

    sd = new ScannerDevice(sanepm, ei, "X5250");

    if (sd == NULL)
    {
        delete sanepm;
        delete ei;
        return SANE_STATUS_INVAL;
    }

    return SANE_STATUS_GOOD;
}

```

---

## Chapter 9 APPLICATION PROGRAMMING INTERFACE

This part provides a detailed description of the Application Programming Interfaces (API's) of each module of the Sample SANE Scanner Driver.

---

### 9.1 Lexmark SANE Backend (x5250/x7170 Backend)

The APIs implemented here are the required APIs by SANE to be implemented by every scanner backend developer except for a few functions (those that are not preceded by sane\_). For more information on the SANE APIs, consult its documentation at <http://www.sane-project.org/docs.html>.

#### 9.1.1 sane\_init ()

##### 9.1.1.1 Description

From the SANE documentation, this is required by SANE to be called before any other SANE function.

##### 9.1.1.2 Prototype

```
SANE_Status sane_init (SANE_Int * version_code,
                      SANE_Auth_Callback authorize);
```

##### 9.1.1.3 Parameters

version\_code – version code of the backend is returned in this parameter  
 authorize – pointer to a function that is invoked when the backend requires authentication for a specific resource.

##### 9.1.1.4 Return Value

Returns SANE\_Status whose value can be any of the following:

SANE_STATUS_GOOD	- Operation completed successfully
SANE_STATUS_GOOD	- Operation completed successfully
SANE_STATUS_UNSUPPORTED	- Operation is not supported.
SANE_STATUS_CANCELLED	- Operation was cancelled.
SANE_STATUS_DEVICE_BUSY	- Device is busy---retry later.
SANE_STATUS_INVALID	- Data or argument is invalid.
SANE_STATUS_EOF	- No more data available (end-of-file).
SANE_STATUS_JAMMED	- Document feeder jammed.
SANE_STATUS_NO_DOCS	- Document feeder out of documents.
SANE_STATUS_COVER_OPEN	- Scanner cover is open.
SANE_STATUS_IO_ERROR	- Error during device I/O.
SANE_STATUS_NO_MEM	- Out of memory.
SANE_STATUS_ACCESS_DENIED	- Access to resource has been denied.

#### 9.1.2 sane\_exit ()

##### 9.1.2.1 Description

From the SANE documentation, this function must be called to terminate the use of a backend.

### 9.1.2.2 Prototype

```
void sane_exit (void);
```

### 9.1.2.3 Parameters

None

### 9.1.2.4 Return Value

None.

## 9.1.3 sane\_get\_devices ()

### 9.1.3.1 Description

From the SANE documentation, this function can be used to query the list of devices that are available.

### 9.1.3.2 Prototype

```
SANE_Status sane_get_devices (const SANE_Device *** device_list,  
                             SANE_Bool local_only);
```

### 9.1.3.3 Parameters

device\_list – stores a pointer to a NULL terminated array of pointers to SANE\_Device structures  
local\_only – Boolean value which determines if only local devices are returned.

### 9.1.3.4 Return Value

Returns SANE\_Status whose value can be any of the following:

SANE_STATUS_GOOD	- Operation completed successfully
SANE_STATUS_UNSUPPORTED	- Operation is not supported.
SANE_STATUS_CANCELLED	- Operation was cancelled.
SANE_STATUS_DEVICE_BUSY	- Device is busy---retry later.
SANE_STATUS_INVALID	- Data or argument is invalid.
SANE_STATUS_EOF	- No more data available (end-of-file).
SANE_STATUS_JAMMED	- Document feeder jammed.
SANE_STATUS_NO_DOCS	- Document feeder out of documents.
SANE_STATUS_COVER_OPEN	- Scanner cover is open.
SANE_STATUS_IO_ERROR	- Error during device I/O.
SANE_STATUS_NO_MEM	- Out of memory.
SANE_STATUS_ACCESS_DENIED	- Access to resource has been denied.

## 9.1.4 sane\_open ()

### 9.1.4.1 Description

From the SANE documentation, this function is used to establish a connection to a particular device.

### 9.1.4.2 Prototype

```
SANE_Status sane_open (SANE_String_Const devicename,  
                      SANE_Handle * handle);
```

### 9.1.4.3 Parameters

devicename – the name of the device to be opened.  
handle – handle to the device opened

### 9.1.4.4 Return Value

Returns SANE\_Status whose value can be any of the following:

SANE_STATUS_GOOD	- Operation completed successfully
SANE_STATUS_UNSUPPORTED	- Operation is not supported.
SANE_STATUS_CANCELLED	- Operation was cancelled.
SANE_STATUS_DEVICE_BUSY	- Device is busy---retry later.
SANE_STATUS_INVALID	- Data or argument is invalid.
SANE_STATUS_EOF	- No more data available (end-of-file).
SANE_STATUS_JAMMED	- Document feeder jammed.
SANE_STATUS_NO_DOCS	- Document feeder out of documents.
SANE_STATUS_COVER_OPEN	- Scanner cover is open.
SANE_STATUS_IO_ERROR	- Error during device I/O.
SANE_STATUS_NO_MEM	- Out of memory.
SANE_STATUS_ACCESS_DENIED	- Access to resource has been denied.

## 9.1.5 sane\_close ()

### 9.1.5.1 Description

From the SANE documentation, this terminates the association between the device handle passed in argument handle and the device it represents.

### 9.1.5.2 Prototype

```
void sane_close (SANE_Handle handle);
```

### 9.1.5.3 Parameters

handle – handle to the device opened.

### 9.1.5.4 Return Value

None.

## 9.1.6 sane\_get\_option\_descriptor ()

### 9.1.6.1 Description

From the SANE documentation, this function is used to access option descriptors.

### 9.1.6.2 Prototype

```
const SANE_Option_Descriptor * sane_get_option_descriptor (
    SANE_Handle handle, SANE_Int option);
```

### 9.1.6.3 Parameters

handle – handle to the device opened.  
option – option number index.

### 9.1.6.4 Return Value

SANE\_Option\_Descriptor – the option descriptor for option number n of the device represented by handle h.

## 9.1.7 sane\_control\_option ()

### 9.1.7.1 Description

From the SANE documentation, this function is used to set or inquire the current value of option number n of the device represented by handle h.

### 9.1.7.2 Prototype

```
SANE_Status sane_control_option (SANE_Handle handle,
                                SANE_Int option, SANE_Action action,
                                void * value, SANE_Int * info);
```

### 9.1.7.3 Parameters

handle - handle to the device opened

option - option number index

action – the way the option is affected by a call to this function is controlled by this parameter, possible values are SANE\_ACTION\_GET\_VALUE, SANE\_ACTION\_SET\_VALUE, SANE\_ACTION\_SET\_AUTO.

### 9.1.7.4 Return Value

Returns SANE\_Status whose value can be any of the following:

SANE_STATUS_GOOD	- Operation completed successfully
SANE_STATUS_UNSUPPORTED	- Operation is not supported.
SANE_STATUS_CANCELLED	- Operation was cancelled.
SANE_STATUS_DEVICE_BUSY	- Device is busy---retry later.
SANE_STATUS_INVALID	- Data or argument is invalid.
SANE_STATUS_EOF	- No more data available (end-of-file).
SANE_STATUS_JAMMED	- Document feeder jammed.
SANE_STATUS_NO_DOCS	- Document feeder out of documents.
SANE_STATUS_COVER_OPEN	- Scanner cover is open.
SANE_STATUS_IO_ERROR	- Error during device I/O.
SANE_STATUS_NO_MEM	- Out of memory.
SANE_STATUS_ACCESS_DENIED	- Access to resource has been denied.

## 9.1.8 sane\_get\_parameters ()

### 9.1.8.1 Description

From the SANE documentation, this function is used to access option descriptors.

### 9.1.8.2 Prototype

```
SANE_Status sane_get_parameters (SANE_Handle handle,
                                SANE_Parameters * params);
```

### 9.1.8.3 Parameters

handle - handle to the device opened

params - structure of type SANE\_Parameters which hold the different scan parameters

### 9.1.8.4 Return Value

Returns SANE\_Status whose value can be any of the following:

SANE_STATUS_GOOD	- Operation completed successfully
SANE_STATUS_UNSUPPORTED	- Operation is not supported.
SANE_STATUS_CANCELLED	- Operation was cancelled.
SANE_STATUS_DEVICE_BUSY	- Device is busy---retry later.
SANE_STATUS_INVALID	- Data or argument is invalid.
SANE_STATUS_EOF	- No more data available (end-of-file).
SANE_STATUS_JAMMED	- Document feeder jammed.
SANE_STATUS_NO_DOCS	- Document feeder out of documents.
SANE_STATUS_COVER_OPEN	- Scanner cover is open.
SANE_STATUS_IO_ERROR	- Error during device I/O.
SANE_STATUS_NO_MEM	- Out of memory.
SANE_STATUS_ACCESS_DENIED	- Access to resource has been denied.

## 9.1.9 sane\_start()

### 9.1.9.1 Description

From the SANE documentation, this function initiates acquisition of an image from the device represented by handle h.

### 9.1.9.2 Prototype

```
SANE_Status sane_start (SANE_Handle handle);
```

### 9.1.9.3 Parameters

handle - handle to the device opened.

### 9.1.9.4 Return Value

Returns SANE\_Status whose value can be any of the following:

SANE_STATUS_GOOD	- Operation completed successfully
SANE_STATUS_UNSUPPORTED	- Operation is not supported.
SANE_STATUS_CANCELLED	- Operation was cancelled.
SANE_STATUS_DEVICE_BUSY	- Device is busy---retry later.
SANE_STATUS_INVALID	- Data or argument is invalid.
SANE_STATUS_EOF	- No more data available (end-of-file).
SANE_STATUS_JAMMED	- Document feeder jammed.
SANE_STATUS_NO_DOCS	- Document feeder out of documents.
SANE_STATUS_COVER_OPEN	- Scanner cover is open.
SANE_STATUS_IO_ERROR	- Error during device I/O.
SANE_STATUS_NO_MEM	- Out of memory.
SANE_STATUS_ACCESS_DENIED	- Access to resource has been denied.

## 9.1.10 sane\_read ()

### 9.1.10.1 Description

From the SANE documentation, this function is used to read image data from the device.

### 9.1.10.2 Prototype

```
SANE_Status sane_read (SANE_Handle handle,
                      SANE_Byte * data, SANE_Int max_length,
                      SANE_Int * length);
```

### 9.1.10.3 Parameters

handle - handle to the device opened  
 data - pointer to a memory area that is at least maxlen bytes long  
 max\_length - length of data  
 length - the number of bytes returned

### 9.1.10.4 Return Value

Returns SANE\_Status whose value can be any of the following:

SANE_STATUS_GOOD	- Operation completed successfully
SANE_STATUS_UNSUPPORTED	- Operation is not supported.
SANE_STATUS_CANCELLED	- Operation was cancelled.
SANE_STATUS_DEVICE_BUSY	- Device is busy---retry later.
SANE_STATUS_INVALID	- Data or argument is invalid.
SANE_STATUS_EOF	- No more data available (end-of-file).
SANE_STATUS_JAMMED	- Document feeder jammed.
SANE_STATUS_NO_DOCS	- Document feeder out of documents.
SANE_STATUS_COVER_OPEN	- Scanner cover is open.
SANE_STATUS_IO_ERROR	- Error during device I/O.
SANE_STATUS_NO_MEM	- Out of memory.
SANE_STATUS_ACCESS_DENIED	- Access to resource has been denied.

## 9.1.11 sane\_cancel()

### 9.1.11.1 Description

From the SANE documentation, this function is used to immediately or as quickly as possible cancel the currently pending operation of the device represented by handle h.

### 9.1.11.2 Prototype

```
void sane_cancel (SANE_Handle handle);
```

### 9.1.11.3 Parameters

handle - handle to the device opened

### 9.1.11.4 Return Value

None

## 9.1.12 sane\_set\_io\_mode()

### 9.1.12.1 Description

From the SANE documentation, this function is used to set the I/O mode of the device handle.

### 9.1.12.2 Prototype

```
SANE_Status sane_set_io_mode (SANE_Handle handle, SANE_Bool non_blocking);
```

### 9.1.12.3 Parameters

handle - handle to the device opened

non\_blocking - Boolean value which determines if I/O mode is blocking or non-blocking

### 9.1.12.4 Return Value

Returns SANE\_Status whose value can be any of the following:

SANE_STATUS_GOOD	- Operation completed successfully
SANE_STATUS_UNSUPPORTED	- Operation is not supported.
SANE_STATUS_CANCELLED	- Operation was cancelled.
SANE_STATUS_DEVICE_BUSY	- Device is busy---retry later.
SANE_STATUS_INVALID	- Data or argument is invalid.
SANE_STATUS_EOF	- No more data available (end-of-file).
SANE_STATUS_JAMMED	- Document feeder jammed.
SANE_STATUS_NO_DOCS	- Document feeder out of documents.
SANE_STATUS_COVER_OPEN	- Scanner cover is open.
SANE_STATUS_IO_ERROR	- Error during device I/O.
SANE_STATUS_NO_MEM	- Out of memory.
SANE_STATUS_ACCESS_DENIED	- Access to resource has been denied.

## 9.1.13 sane\_get\_select\_fd()

### 9.1.13.1 Description

From the SANE documentation, this function is used to obtain a (platform-specific) file-descriptor for handle that is readable if and only if the image data is available.

### 9.1.13.2 Prototype

```
SANE_Status sane_get_select_fd (SANE_Handle handle, SANE_Int * fd);
```

### 9.1.13.3 Parameters

handle - handle to the device opened

fd - selected file descriptor

### 9.1.13.4 Return Value

Returns SANE\_Status whose value can be any of the following:

SANE_STATUS_GOOD	- Operation completed successfully
SANE_STATUS_UNSUPPORTED	- Operation is not supported.
SANE_STATUS_CANCELLED	- Operation was cancelled.
SANE_STATUS_DEVICE_BUSY	- Device is busy---retry later.
SANE_STATUS_INVALID	- Data or argument is invalid.

---

SANE_STATUS_EOF	- No more data available (end-of-file).
SANE_STATUS_JAMMED	- Document feeder jammed.
SANE_STATUS_NO_DOCS	- Document feeder out of documents.
SANE_STATUS_COVER_OPEN	- Scanner cover is open.
SANE_STATUS_IO_ERROR	- Error during device I/O.
SANE_STATUS_NO_MEM	- Out of memory.
SANE_STATUS_ACCESS_DENIED	- Access to resource has been denied.

## 9.1.14 init\_options()

### 9.1.14.1 Description

This function is used to initialize the SANE option that describes the capabilities of a Lexmark scanner device.

### 9.1.14.2 Prototype

```
SANE_Status init_options(Lexmark_Device * lexmark_device);
```

### 9.1.14.3 Parameters

lexmark\_device – pointer to a structure which describes the options of the device, handle to a Lexmark scanner device, pointer to the next Lexmark scanner device, SANE parameters and state of the device.

### 9.1.14.4 Return Value

Returns SANE\_Status whose value can be any of the following:

SANE_STATUS_GOOD	- Operation completed successfully
SANE_STATUS_UNSUPPORTED	- Operation is not supported.
SANE_STATUS_CANCELLED	- Operation was cancelled.
SANE_STATUS_DEVICE_BUSY	- Device is busy---retry later.
SANE_STATUS_INVALID	- Data or argument is invalid.
SANE_STATUS_EOF	- No more data available (end-of-file).
SANE_STATUS_JAMMED	- Document feeder jammed.
SANE_STATUS_NO_DOCS	- Document feeder out of documents.
SANE_STATUS_COVER_OPEN	- Scanner cover is open.
SANE_STATUS_IO_ERROR	- Error during device I/O.
SANE_STATUS_NO_MEM	- Out of memory.
SANE_STATUS_ACCESS_DENIED	- Access to resource has been denied.

## 9.1.15 attachLexmark()

### 9.1.15.1 Description

Callback function that creates an array of Lexmark device supported by a particular frontend.

### 9.1.15.2 Prototype

```
SANE_Status attachLexmark (SANE_String_Const devname);
```

### 9.1.15.3 Parameters

devname – the name of the device to be opened.

### 9.1.15.4 Return Value

Returns SANE\_Status whose value can be any of the following:

SANE_STATUS_GOOD	- Operation completed successfully
SANE_STATUS_UNSUPPORTED	- Operation is not supported.
SANE_STATUS_CANCELLED	- Operation was cancelled.
SANE_STATUS_DEVICE_BUSY	- Device is busy---retry later.
SANE_STATUS_INVALID	- Data or argument is invalid.
SANE_STATUS_EOF	- No more data available (end-of-file).
SANE_STATUS_JAMMED	- Document feeder jammed.
SANE_STATUS_NO_DOCS	- Document feeder out of documents.
SANE_STATUS_COVER_OPEN	- Scanner cover is open.
SANE_STATUS_IO_ERROR	- Error during device I/O.
SANE_STATUS_NO_MEM	- Out of memory.
SANE_STATUS_ACCESS_DENIED	- Access to resource has been denied.

---

## 9.2 SanePortMonitor

### 9.2.1 SanePortMonitor::SanePortMonitor()

#### 9.2.1.1 Description

The constructor of the Linux-specific implementation of the PortMonitor.

#### 9.2.1.2 Prototype

```
SanePortMonitor::SanePortMonitor();
```

#### 9.2.1.3 Parameters

None

#### 9.2.1.4 Return Value

None

### 9.2.2 SanePortMonitor::~~SanePortMonitor()

#### 9.2.2.1 Description

The destructor of the Linux-specific implementation of the PortMonitor.

#### 9.2.2.2 Prototype

```
SanePortMonitor::~~SanePortMonitor();
```

#### 9.2.2.3 Parameters

None

#### 9.2.2.4 Return Value

None

## 9.2.3 SanePortMonitor::PM\_Open()

### 9.2.3.1 Description

Opens the communication port.

### 9.2.3.2 Prototype

```
PortMonitor::PM_Error SanePortMonitor::PM_Open();
```

### 9.2.3.3 Parameters

None

### 9.2.3.4 Return Value

Returns a PortMonitor:PM\_Error type:

- a.) PortMonitor::PM\_ERR\_SUCCESS if call is successful.
- b.) PortMonitor::PM\_ERR\_FAILED is call failed.

## 9.2.4 SanePortMonitor::PM\_Close()

### 9.2.4.1 Description

Closes the communication port.

### 9.2.4.2 Prototype

```
PortMonitor::PM_Error SanePortMonitor::PM_Close();
```

### 9.2.4.3 Parameters

None

### 9.2.4.4 Return Value

Returns a PortMonitor:PM\_Error type:

- a.) PortMonitor::PM\_ERR\_SUCCESS if call is successful.
- b.) PortMonitor::PM\_ERR\_FAILED is call failed.

## 9.2.5 SanePortMonitor::PM\_Write()

### 9.2.5.1 Description

Sends data or information to the Lexmark scanner device.

### 9.2.5.2 Prototype

```
long SanePortMonitor::PM_Write(unsigned char * buffer, unsigned long size);
```

### 9.2.5.3 Parameters

buffer – This contains the data to be written to the scanner port.  
size – the number of bytes to be written.

### 9.2.5.4 Return Value

The actual number of bytes written, or -1 if a timeout or an error occurred during the write call.

## 9.2.6 SanePortMonitor::PM\_Read()

### 9.2.6.1 Description

Reads data or information from the Lexmark scanner device.

### 9.2.6.2 Prototype

```
long LinuxPortMonitor::PM_Read(unsigned char * buffer, unsigned long size);
```

### 9.2.6.3 Parameters

buffer – This will contain the data to be read from the scanner port.  
size – the number of bytes to be read.

### 9.2.6.4 Return Value

The actual number of bytes read, or -1 if a timeout or an error occurred during the write call.

## 9.2.7 SanePortMonitor::PM\_SetDeviceName()

### 9.2.7.1 Description

Sets the device name to be opened.

### 9.2.7.2 Prototype

```
PortMonitor::PM_Error SanePortMonitor::PM_SetDeviceName(  
    SANE_String_Const devname);
```

### 9.2.7.3 Parameters

devname – device to be opened

### 9.2.7.4 Return Value

Returns a PortMonitor:PM\_Error type:  
a.) PortMonitor::PM\_ERR\_SUCCESS if call is successful.  
b.) PortMonitor::PM\_ERR\_FAILED is call failed.

## 9.2.8 SanePortMonitor::PM\_SetDeviceName()

### 9.2.8.1 Description

Sets the device name to be opened.

### 9.2.8.2 Prototype

```
PortMonitor::PM_Error SanePortMonitor::PM_SetDeviceName(  
    SANE_String_Const devname);
```

### 9.2.8.3 Parameters

devname – device to be opened

### 9.2.8.4 Return Value

Returns a PortMonitor:PM\_Error type:

- a.) PortMonitor::PM\_ERR\_SUCCESS if call is successful.
- b.) PortMonitor::PM\_ERR\_FAILED is call failed.

## 9.2.9 SanePortMonitor::PM\_AttachDevice()

### 9.2.9.1 Description

Routine to attach a scanner device to the backend.

### 9.2.9.2 Prototype

```
PortMonitor::PM_Error SanePortMonitor::PM_AttachDevice(  
    const char * name,  
    SANE_Status (* attach) (const char * dev));
```

### 9.2.9.3 Parameters

name – name of the device

attach – pointer to a function which implements the attach routine

### 9.2.9.4 Return Value

Returns a PortMonitor:PM\_Error type:

- a.) PortMonitor::PM\_ERR\_SUCCESS if call is successful.
- b.) PortMonitor::PM\_ERR\_FAILED is call failed.

---

## 9.3 ScanErrorCommunicator

### 9.3.1 ScanErrorCommunicator ::ScanErrorCommunicator()

#### 9.3.1.1 Description

The constructor of the Linux-specific implementation of the ErrorCommunicator.

#### 9.3.1.2 Prototype

```
ScanErrorCommunicator::ScanErrorCommunicator();
```

#### 9.3.1.3 Parameters

None

#### 9.3.1.4 Return Value

None

---

## 9.3.2 ScanErrorCommunicator::~ScanErrorCommunicator()

### 9.3.2.1 Description

The destructor of the Linux-specific implementation of the ErrorCommunicator.

### 9.3.2.2 Prototype

```
ScanErrorCommunicator::~ScanErrorCommunicator();
```

### 9.3.2.3 Parameters

None

### 9.3.2.4 Return Value

None

## 9.3.3 ScanErrorCommunicator::ReportError()

### 9.3.3.1 Description

Writes the error encountered to the scanner log file.

### 9.3.3.2 Prototype

```
void ScanErrorCommunicator::ReportError(  
    ScanErrorInterface::ScannerError error);
```

### 9.3.3.3 Parameters

error – The code representing the error encountered by the scanner device.

### 9.3.3.4 Return Value

None

## 9.3.4 ScanErrorCommunicator::GetUserDecision()

### 9.3.4.1 Description

Returns the user response regarding whether to continue or to abort a scan job after it encountered a non-critical error. In case of scanning, the backend this API just returns a CONTINUE decision.

### 9.3.4.2 Prototype

```
ScanErrorInterface::UserDecision ScanErrorCommunicator::GetUserDecision();
```

### 9.3.4.3 Parameters

None

### 9.3.4.4 Return Value

None

## **9.4 LEXMARK LINUX PRINTER DRIVER DEVELOPER'S KIT**

Please refer to the LDK Interface Specifications document for more information.