

# **plxeprom**

**PLX Device EEPROM Access Software**

## **plxeprom**

# **Linux Device Driver, API Library and Software User Manual**

**Manual Revision: August 20, 2025  
Driver Release Version 1.2.115.52.0**

**General Standards Corporation  
8302A Whitesburg Drive  
Huntsville, AL 35802  
Phone: (256) 880-8787  
Fax: (256) 880-8788**

**URL: <http://www.generalstandards.com>**

**E-mail: [sales@generalstandards.com](mailto:sales@generalstandards.com)**

**E-mail: [support@generalstandards.com](mailto:support@generalstandards.com)**

## Preface

Copyright © 2019-2025, **General Standards Corporation**

Additional copies of this manual or other literature may be obtained from:

**General Standards Corporation**

8302A Whitesburg Dr.

Huntsville, Alabama 35802

Phone: (256) 880-8787

FAX: (256) 880-8788

URL: <http://www.generalstandards.com>

E-mail: [sales@generalstandards.com](mailto:sales@generalstandards.com)

**General Standards Corporation** makes no warranty of any kind with regard to this documentation and/or software, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Although extensive editing and reviews are performed before release, **General Standards Corporation** assumes no responsibility for any errors, inaccuracies or omissions herein. This documentation, information and software are made available solely on an “as-is” basis. Nor is there any commitment to update or keep current this documentation.

**General Standards Corporation** does not assume any liability arising out of the application or use of documentation, software, product or circuit described herein, nor is any license conveyed under any patent rights or any rights of others.

**General Standards Corporation** assumes no responsibility for any consequences resulting from omissions or errors in this manual or from the use of information contained herein.

**General Standards Corporation** reserves the right to make any changes, without notice, to this documentation, software or product, to improve accuracy, clarity, reliability, performance, function, or design.

ALL RIGHTS RESERVED.

GSC is a trademark of **General Standards Corporation**.

PLX and PLX Technology are trademarks of PLX Technology, Inc.

# Table of Contents

<b>1. Introduction.....</b>	<b>7</b>
1.1. Purpose.....	7
1.2. Acronyms.....	7
1.3. Definitions .....	7
1.4. Software Overview .....	8
1.4.1. Basic Software Architecture .....	8
1.4.2. API Library.....	8
1.4.3. Device Driver .....	8
1.5. Hardware Overview .....	9
1.6. Reference Material.....	9
1.7. Licensing.....	9
<b>2. Installation .....</b>	<b>10</b>
2.1. CPU and Kernel Support.....	10
2.1.1. 32-bit Support Under 64-bit Environments .....	11
2.2. The /proc/ File System .....	11
2.3. File List.....	11
2.4. Directory Structure.....	11
2.5. Installation .....	12
2.6. Removal.....	12
2.7. Overall Make Script.....	13
2.8. Environment Variables .....	13
2.8.1. GSC_API_COMP_FLAGS.....	13
2.8.2. GSC_API_LINK_FLAGS.....	13
2.8.3. GSC_LIB_COMP_FLAGS.....	14
2.8.4. GSC_LIB_LINK_FLAGS.....	14
2.8.5. GSC_APP_COMP_FLAGS.....	14
2.8.6. GSC_APP_LINK_FLAGS.....	14
<b>3. Main Interface Files.....</b>	<b>15</b>
3.1. Main Header File .....	15
3.2. Main Library File.....	15
3.2.1. Build .....	15
3.2.2. System Libraries.....	16
3.2.3. Shared Object Script: Build the Main Libraries as Shared Object Files.....	16
<b>4. API Library .....</b>	<b>17</b>
4.1. Files.....	17
4.2. Build .....	17
4.3. Library Use .....	17
4.4. Macros .....	17

4.4.1. EEPROM Register Identifier .....	18
4.4.2. IOCTL Services .....	18
4.4.3. Registers .....	18
4.5. Data Types .....	19
4.6. Functions .....	19
4.6.1. plxeprom_close() .....	19
4.6.2. plxeprom_init() .....	20
4.6.3. plxeprom_ioctl() .....	20
4.6.4. plxeprom_open() .....	21
4.6.5. plxeprom_read() .....	22
4.7. IOCTL Services .....	23
4.7.1. PLXEEPROM_IOCTL_INITIALIZE .....	23
4.7.2. PLXEEPROM_IOCTL_QUERY .....	23
4.7.3. PLXEEPROM_IOCTL_REG_MOD .....	25
4.7.4. PLXEEPROM_IOCTL_REG_READ .....	25
4.7.5. PLXEEPROM_IOCTL_REG_WRITE .....	26
<b>5. The Driver .....</b>	<b>28</b>
5.1. Files .....	28
5.2. Build .....	28
5.3. Startup .....	28
5.3.1. Manual Driver Startup Procedures .....	28
5.3.2. Automatic Driver Startup Procedures .....	29
5.4. Verification .....	30
5.5. Version .....	31
5.6. Shutdown .....	31
<b>6. Document Source Code Examples .....</b>	<b>32</b>
6.1. Files .....	32
6.2. Build .....	32
6.3. Library Use .....	32
<b>7. Utilities Source Code .....</b>	<b>33</b>
7.1. Files .....	33
7.2. Build .....	33
7.3. Library Use .....	33
<b>8. Operating Information .....</b>	<b>34</b>
8.1. Debugging Aids .....	34
8.1.1. Device Identification .....	34
8.2. PLX EEPROM Support .....	34
8.3. PLX EEPROM Organization .....	34
8.4. plxeprom Software Utility .....	35
8.4.1. Identify What Is Installed .....	35
8.4.2. Backup The EEPROM Configuration Image .....	36

8.4.3. In Case of An Error, Restore the EEPROM Configuration Image .....	36
8.4.4. Device Identity .....	36
8.4.5. Vendor ID .....	36
8.4.6. Device ID .....	37
8.4.7. Class Code .....	37
8.4.8. BAR1 Configuration .....	37
8.4.9. BAR2 Configuration .....	37
8.4.10. Load An Existing .EEP File .....	37
<b>9. Sample Applications .....</b>	<b>39</b>
9.1. id - Identify Board - ../id/ .....	39
9.2. plxeprom – PLX EEPROM Access - ../plxeprom/ .....	39
9.3. regs - Register Access - ../regs/ .....	39
<b>Document History .....</b>	<b>40</b>

Table of Figures

Figure 1 Basic architectural representation.....8

# 1. Introduction

The plxeprom device driver was written with the goal of providing a tool for modifying a few common EEPROM based configuration settings on PLX based PCI devices. Example settings include the PCI Class Code and mapping of BAR1 and BAR2. With this tool, Linux customers can much more easily make such configuration changes in the field. Otherwise, the alternatives have been either sending boards back to General Standards or locate a Windows host upon which to install PLXMON.

Most General Standards drivers and accompanying documentation are written for those who have purchased General Standards' I/O boards and intend to use those boards as a central part of their own applications. While the plxeprom driver can be used in the same manner, it does not provide access to the full capabilities of the purchased I/O boards. Instead, this driver provides access only to the PLX configuration EEPROM, the PCI registers and the PLX registers. Though the focus of this driver is very restrictive compared to device specific drivers, this manual is written in the same format.

The plxeprom software utility is the central focus of the plxeprom driver archive. The device driver provides the means of communicating directly with the hardware and is where the functionality exists for reading from and writing to the EEPROM. The API Library provides the structured interface that software uses to send requests to the driver. But the plxeprom application is where the smarts reside regarding which EEPROM bits and bytes to modify in order to change the PLX configuration. For usage information refer to section 8.4 (page 35).

## 1.1. Purpose

The purpose of this document is to describe the interface to the plxeprom API Library and to the underlying Linux device driver. The API Library software provides the interface between "Application Software" and the device driver. The driver software provides the interface between the API Library and installed PLX based I/O cards. The API Library and driver interfaces are designed specifically to provide access to PCI registers and the installed PLX configuration EEPROM.

The purpose of this document is also to describe the primary software tool that facilitates access to the PLX configuration EEPROM included with General Standards' PCI based products. This tool consists of the plxeprom sample application. For additional information refer to section 8.4 (page 35).

## 1.2. Acronyms

The following is a list of commonly occurring acronyms which may appear throughout this document.

Acronyms	Description
CC	Class Code
DID	Device ID
GSC	General Standards Corporation
EEPROM	Electrically Erasable Programmable Read-Only Memory
SID	Subsystem ID
SVID	Subsystem Vendor ID
VID	Vendor ID

## 1.3. Definitions

The following is a list of commonly occurring terms which may appear throughout this document.

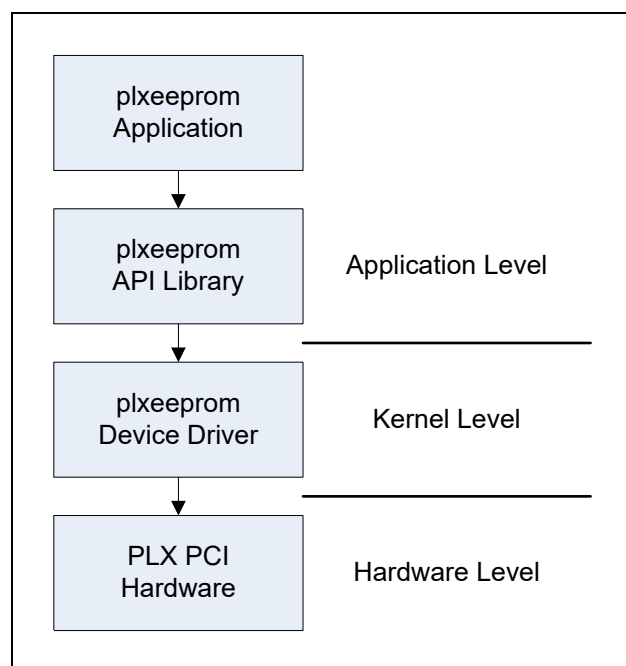
Term	Definition
...	This is a shortcut representation of the plxeprom installation directory or any of its subdirectories.
API Library	This refers to the library that provides user level access to the plxeprom device driver.

Application	This refers to the user mode process, which runs in user space with user mode privileges.
Driver	This refers to the kernel mode device driver, which runs in kernel space with kernel mode privileges.
plxeprom	This is a general reference to the device driver, the API Library or the plxeprom sample application.

## 1.4. Software Overview

### 1.4.1. Basic Software Architecture

This section describes the general architecture for the basic components that comprise plxeprom applications. The overall architecture is illustrated in Figure 1 below.



**Figure 1** Basic architectural representation.

### 1.4.2. API Library

The primary means of accessing plxeprom detected boards is via the plxeprom API Library. This library forms a layer between the application and the driver. Additional information is given in in section 4 (page 17). With the library, applications are able to open and close a device and, while open, perform I/O control operations. The I/O control operations are primarily aimed at reading and writing the board's configuration EEPROM. (See Figure 1 above.)

### 1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with devices using PLX based PCI bridge chips. The driver executes under control of the operating system and runs in Kernel Mode as a Kernel Mode device driver. The driver is implemented as a standard dynamically loadable Linux device driver written in the C programming language. While applications can access the driver directly without use of the API Library, it is recommended that all access be made through the Library. (See Figure 1 above.)



## 1.5. Hardware Overview

For an overview of the board hardware refer to this same section number in the OS specific driver user manual for the board being accessed.

## 1.6. Reference Material

The following reference material may be of particular benefit in using the plxeprom software. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on boards using a PLX PCI bridge chip.

- The *PCI9056 PCI Bridge Chip* data handbook from PLX Technology, Inc. \*
- The *PCI9060 PCI Bridge Chip* data handbook from PLX Technology, Inc. \*
- The *PCI9080 PCI Bridge Chip* data handbook from PLX Technology, Inc. \*
- The *PCI9656 PCI Bridge Chip* data handbook from PLX Technology, Inc. \*

\* The PLX material is available from the following source.

PLX Technology Inc.  
870 Maude Avenue  
Sunnyvale, California 94085 USA  
Phone: 1-800-759-3735  
WEB: <http://www.plxtech.com>

## 1.7. Licensing

For licensing information please refer to the text file `LICENSE.txt` in the root installation directory.

## 2. Installation

### 2.1. CPU and Kernel Support

The driver is designed to operate with Linux kernel versions 6.x, 5.x, 4.x, 3.x, 2.6, 2.4 and 2.2 running on a PC system with one or more x86 processors. This release of the driver supports the below listed kernels.

Kernel	Distribution
6.8.5	Red Hat Fedora Core 40
6.5.6	Red Hat Fedora Core 39
6.2.9	Red Hat Fedora Core 38
6.0.7	Red Hat Fedora Core 37
5.17.5	Red Hat Fedora Core 36
5.14.10	Red Hat Fedora Core 35
5.11.12	Red Hat Fedora Core 34
5.8.15	Red Hat Fedora Core 33
5.6.6	Red Hat Fedora Core 32
5.3.7	Red Hat Fedora Core 31
5.0.9	Red Hat Fedora Core 30
4.18.16	Red Hat Fedora Core 29
4.16.3	Red Hat Fedora Core 28
4.13.9	Red Hat Fedora Core 27
4.11.8	Red Hat Fedora Core 26
4.8.6	Red Hat Fedora Core 25
4.5.5	Red Hat Fedora Core 24
4.2.3	Red Hat Fedora Core 23
4.0.4	Red Hat Fedora Core 22
3.17.4	Red Hat Fedora Core 21
3.11.10	Red Hat Fedora Core 20
3.9.5	Red Hat Fedora Core 19
3.6.10	Red Hat Fedora Core 18
3.3.4	Red Hat Fedora Core 17
3.1.0	Red Hat Fedora Core 16
2.6.38	Red Hat Fedora Core 15
2.6.35	Red Hat Fedora Core 14
2.6.33	Red Hat Fedora Core 13
2.6.31	Red Hat Fedora Core 12
2.6.29	Red Hat Fedora Core 11
2.6.27	Red Hat Fedora Core 10
2.6.25	Red Hat Fedora Core 9
2.6.23	Red Hat Fedora Core 8
2.6.21	Red Hat Fedora Core 7
2.6.18	Red Hat Fedora Core 6
2.6.15	Red Hat Fedora Core 5
2.6.11	Red Hat Fedora Core 4
2.6.9	Red Hat Fedora Core 3
2.4.18	Red Hat 8.0

**NOTE:** Some older kernel versions are supported (the sources are maintained), but are not tested.

**NOTE:** While only Red Hat Fedora distributions are listed, numerous other distributions are supported and have been tested on an as needed basis.

**NOTE:** The driver will have to be built before being used as it is provided in source form only.

**NOTE:** The driver has not been tested with a non-versioned kernel.

**NOTE:** The driver is designed for SMP support, but has not undergone SMP specific testing.

### 2.1.1. 32-bit Support Under 64-bit Environments

This driver supports 32-bit applications under 64-bit environments. The availability of this feature in the kernel depends on a 64-bit kernel being configured to support 32-bit application compatibility. Additionally, 2.6 kernels prior to 2.6.11 implemented 32-bit compatibility in a way that resulted in some drivers not being able to take advantage of the feature. (In these kernels a driver's IOCTL command codes must be globally unique. Beginning with 2.6.11 this requirement has been lifted.) If the driver is not able to provide 32-bit support under a 64-bit kernel, the "32-bit support" field in the `/proc/plxeprom` file will be "no".

## 2.2. The `/proc/` File System

While the driver is running, the text file `/proc/plxeprom` can be read to obtain information about the driver and the boards it detects. Each file entry includes an entry name followed immediately by a colon, a space character, and the entry value. Below is an example of what appears in the file, followed by descriptions of each entry.

```
version: 1.2.116.52
32-bit support: yes
boards: 1
models: 16A016
ids: 0x9056
```

Entry	Description
version	This gives the driver version number in the form <code>x.x.x.x</code> .
32-bit support	This reports the driver's support for 32-bit applications. This will be either "yes" or "no" for 64-bit driver builds and "yes (native)" for 32-bit builds.
boards	This identifies the total number of boards the driver detected.
models	This gives a comma separated list of the model number for each board the driver detected. The plxeprom driver may not be able to identify installed boards as accurately as their dedicated drivers. As a result, some model numbers may be given as "GSC_0xXXXX" or "OTHER_0xXXXX", where the "XXXX" reflects the hard coded Device ID value.
ids	This gives a comma separated list of the PLX chip model number for each board the driver detected.

## 2.3. File List

This release consists of the below listed primary files. The archive content is described in following subsections.

File	Description
<code>plxeprom.linux.tar.gz</code>	This archive contains the driver, the API Library and all related files.
<code>plxeprom_linux_um.pdf</code>	This is a PDF version of this user manual, which is included in the archive.

## 2.4. Directory Structure

The following table describes the directory structure utilized by the installed files. During installation the directory structure is created and populated with the respective files.

Directory	Description
plxeprom/	This is the driver root directory. It contains the documentation, the Overall Make Script (section 2.7, page 13) and the below listed subdirectories.
.../api/	This directory contains the API Library source files (section 4, page 17).
.../docsrc/	This directory contains the source files for the code samples given in this document (section 6, page 32).
.../driver/	This directory contains the device driver source files (section 5, page 28).
.../include/	This directory contains the header files for the various libraries.
.../lib/	This directory contains all of the libraries built from the installed sources.
.../samples/	This directory contains the sample application subdirectories and all of their corresponding source files (section 9, page 39).
.../utils/	This directory contains the source files for the utility libraries used by the sample applications (section 7, page 33).

## 2.5. Installation

Perform installation following the below listed steps. This installs the device driver, the API Library and all related sources and documentation.

1. Create and change to the directory where the files are to be installed, such as `/usr/src/linux/drivers/`. (The path name may vary among distributions and kernel versions.)
2. Copy the archive file `plxeprom.linux.tar.gz` into the current directory.
3. Issue the following command to decompress and extract the files from the provided archive. This creates the directory `plxeprom` in the current directory, and then copies all of the archive's files into this new directory.

```
tar -xzvf plxeprom.linux.tar.gz
```

## 2.6. Removal

Perform removal following the below listed steps. This removes the device driver, the API Library and all related sources and documentation.

**NOTE:** The following steps may require elevated privileges.

1. Shutdown the driver as described in section 5.6 (page 31).
2. Change to the directory where the driver archive was installed, which may have been `/usr/src/linux/drivers/`. (The path name may vary among distributions and kernel versions.)
3. Issue the below command to remove the driver archive and all of the installed driver files.

```
rm -rf plxeprom.linux.tar.gz plxeprom
```

4. Issue the below command to remove all of the installed device nodes.

```
rm -f /dev/plxeprom.*
```

5. If the automatic startup procedure was adopted (section 5.3.2, page 29), then edit the system startup script `rc.local` and remove the line that invokes the `plxeprom`'s start script. The file `rc.local` should be located in the `/etc/rc.d/` directory.

## 2.7. Overall Make Script

An Overall Make Script is included in the root installation directory. Executing this script will perform a make for all build targets included in the release. The script also loads the driver and copies the API Library to `/usr/lib/`. The script is named `make_all`. Follow the below steps to perform an overall make and to load the driver.

**NOTE:** The following steps may require elevated privileges.

1. Change to the driver root directory (`.../plxeeprom/`).
2. Remove existing build targets using the below command. This does not unload the driver.

```
./make_all clean
```

3. Issue the following command to make all archive targets and to load the driver.

```
./make_all
```

## 2.8. Environment Variables

Some build environments may require compiler or linker options not present in the provided make files. To accommodate local environment specific requirements, the provided make files incorporate support for the following set of GSC specific environment variables.

### 2.8.1. GSC\_API\_COMP\_FLAGS

This environment variable accommodates adding compiler command line options when compiling source files for the API Library. The compiler used by the API Library make file is “`gcc`”. The content of this environment variable is noted in the make file’s output to the screen. The table below shows a portion of the screen output. The “xxx” in the table refers to the contents of the environment variable. This environment variable has no effect on compiling any other distributed source files or linking of any object files.

<b>Undefined or Empty</b>	== Compiling: <code>init.c</code>	
	== Compiling: <code>ioctl.c</code>	
	== Compiling: <code>open.c</code>	
<b>Defined and Not Empty</b>	== Compiling: <code>init.c</code> (added 'xxx')	
	== Compiling: <code>ioctl.c</code> (added 'xxx')	
	== Compiling: <code>open.c</code> (added 'xxx')	

### 2.8.2. GSC\_API\_LINK\_FLAGS

This environment variable accommodates adding linker command line options when linking object files for the API Library. The linker used by the API Library make file is “`ld`”. The content of this environment variable is noted in the make file’s output to the screen. The table below shows a portion of the screen output. The “xxx” in the table refers to the contents of the environment variable. This environment variable has no effect on compiling of any source files or linking of any other object files.

<b>Undefined or Empty</b>	==== Linking: <code>../lib/libplxeeprom_api.so</code>
<b>Defined and Not Empty</b>	==== Linking: <code>../lib/libplxeeprom_api.so</code> (added 'xxx')

### 2.8.3. GSC\_LIB\_COMP\_FLAGS

This environment variable accommodates adding compiler command line options when compiling source files for the utility libraries. The compiler used by the utility library make files is “gcc”. The content of this environment variable is noted in the make files’ output to the screen. The table below shows a portion of the screen output. The “xxx” in the table refers to the contents of the environment variable. This environment variable has no effect on compiling any other distributed source files or linking of any object files.

<b>Undefined or Empty</b>	== Compiling: close.c == Compiling: init.c == Compiling: ioctl.c
<b>Defined and Not Empty</b>	== Compiling: close.c (added 'xxx') == Compiling: init.c (added 'xxx') == Compiling: ioctl.c (added 'xxx')

### 2.8.4. GSC\_LIB\_LINK\_FLAGS

This environment variable accommodates adding linker command line options when linking object files for the utility libraries. The linker used by the utility library make files is “ld”. The content of this environment variable is noted in the make files’ output to the screen. The table below shows a portion of the screen output. The “xxx” in the table refers to the contents of the environment variable. This environment variable has no effect on compiling of any source files or linking of any other object files.

<b>Undefined or Empty</b>	==== Linking: ../lib/plxeprom_utils.a
<b>Defined and Not Empty</b>	==== Linking: ../lib/plxeprom_utils.a (added 'xxx')

### 2.8.5. GSC\_APP\_COMP\_FLAGS

This environment variable accommodates adding compiler command line options when compiling source files for the sample applications. The compiler used by the sample application make files is “gcc”. The content of this environment variable is noted in the make files’ output to the screen. The table below shows a portion of the screen output. The “xxx” in the table refers to the contents of the environment variable. This environment variable has no effect on compiling any other distributed source files or linking of any object files.

<b>Undefined or Empty</b>	== Compiling: main.c == Compiling: perform.c
<b>Defined and Not Empty</b>	== Compiling: main.c (added 'xxx') == Compiling: perform.c (added 'xxx')

### 2.8.6. GSC\_APP\_LINK\_FLAGS

This environment variable accommodates adding linker command line options when linking object files for the sample applications. The linker used by the sample application make files is “gcc”. The content of this environment variable is noted in the make files’ output to the screen. The table below shows a portion of the screen output. The “xxx” in the table refers to the contents of the environment variable. This environment variable has no effect on compiling of any source files or linking of any other object files.

<b>Undefined or Empty</b>	==== Linking: id
<b>Defined and Not Empty</b>	==== Linking: id (added 'xxx')

### 3. Main Interface Files

This section gives general information on the suggested device interface files to use when developing plxeeprom based applications.

#### 3.1. Main Header File

Throughout the remainder of this document references are made to various header files included as part of the plxeeprom driver installation. For ease of use it is suggested that applications include only the single header file shown below rather than individually including those headers identified separately later in this document. Including this header file pulls in all other pertinent plxeeprom specific header files. Therefore, sources may include only this one plxeeprom header and make files may reference only this one plxeeprom include directory.

Description	File	Location
Header File	plxeeprom_main.h	.../include/

#### 3.2. Main Library File

Throughout the remainder of this document references are made to various statically linkable libraries included as part of the plxeeprom driver installation. For ease of use it is suggested that applications link only the single library file shown below rather than individually linking those libraries identified separately later in this document. Linking this library file pulls in all other static libraries included with the driver. Therefore, make files may reference only this one plxeeprom static library and only this one plxeeprom library directory.

Description	File	Location
Static Library	plxeeprom_main.a	.../lib/
	plxeeprom_multi.a	

**NOTE:** For applications using the plxeeprom and no other GSC devices, link the plxeeprom\_main.a library. For applications using multiple GSC device types, link the xxxx\_main.a library for one of the devices and the xxxx\_multi.a library for the others. Linking multiple xxxx\_main.a libraries may likely produce link errors due to duplicate symbols being defined. While it may make little or no difference, it is recommended that one choose the xxxx\_main.a library from the driver with the largest number in positions three (x.x.X.x.x) and/or four (x.x.x.X.x) in the driver release version number.

**NOTE:** The plxeeprom API Library is implemented as a shared library and is thus not linked with the plxeeprom Main Library. The API Library must be linked with applications by adding the argument `-lplxeeprom_api` to the linker command line.

##### 3.2.1. Build

The main library is built via the Overall Make Script (section 2.7, page 13). However, the main library can be built separately following the below steps.

1. Change to the directory where the main library resides (.../lib/).
2. Remove existing build targets using the below command.

```
make clean
```

3. Build the main library by issuing the below command.

```
make
```

### 3.2.2. System Libraries

In addition to linking the static library named above, as well as the API Library shared object file, applications may need to also link in additional system libraries as noted below.

Library	gcc Link Flag
Math	-lm
POSIX Thread	-lpthread
Real Time	-lrt

### 3.2.3. Shared Object Script: Build the Main Libraries as Shared Object Files

The main libraries built via the Overall Make Script (section 2.7, page 13) are static library files. Some applications however, require that the Main Libraries be accessed as shared object files. Generating shared object files require that all of the static libraries be recompiled for this purpose and linked as .so files. This is done using the Shared Object Script named below. When run, the script invokes the Overall Make Script to clean all existing build targets, deletes the two shared object files named below, if they exist, defines an environment variable used by all of the static library make files, invokes the Overall Make Script again to rebuild all existing build targets then invokes make on the library make file (.../lib/makefile) to link the shared object files. The required manual steps are as follows.

1. Change to the directory where the main library files reside (.../lib/).
2. Execute the below script.

```
./static_to_shared.sh
```

Running the above-named Shared Object Script produces the files given in the table below. These shared object files fulfill the same purpose as the similarly named static libraries as described in the note under section 3.2 above. Refer to that note when selecting which shared object file to use.

Description	File	Location
Shared Object Files	libplxeprom_main.so libplxeprom_multi.so libplxeprom_all.so†	.../lib/

† This library includes all generated libraries, including the API Library shared object file content.

The shared object files can be linked via two different methods. In the first method, the application linker command line can explicitly name the file in the same manner as is done were it a static library. This is the method used by the sample applications, all of which use the plxeprom API Library, which itself is a shared object file. This file is also found in the .../lib/ subdirectory. In the second method, the .so files are copied to the /usr/lib/ subdirectory and are referenced on the application's linker command line as given in the table below.

Library	gcc Link Flag
libplxeprom_main.so	-lplxeprom_main
libplxeprom_multi.so	-lplxeprom_multi
libplxeprom_all.so†	-lplxeprom_all

† This library includes all generated libraries, including the API Library shared object file content.



## 4. API Library

The plxeeprom API Library is the software interface between user applications and the plxeeprom device driver. The interface is accessed by including the header file `plxeeprom_api.h`.

**NOTE:** Contact General Standards Corporation if additional library functionality is required.

### 4.1. Files

The library files are summarized in the table below.

Description	File	Location
Source Files	*.c, *.h ...	.../api/
Header File	plxeeprom_api.h	.../include/
Library File	libplxeeprom_api.so	.../lib/ /usr/lib/ †

† The shared object library is automatically copied to `/usr/lib/` when it is built.

### 4.2. Build

The API Library is built via the Overall Make Script (section 2.7, page 13), but can be built separately following the below steps.

**NOTE:** The following steps may require elevated privileges.

1. Change to the directory where the library sources are installed (`.../api/`).
2. Remove existing build targets using the below command.

```
make clean
```

3. Compile the source files and build the library by issuing the below command. This step copies the API Library file to `/usr/lib/`.

```
make
```

### 4.3. Library Use

The library is used at application compile time, at application link time and at application run time. At compile time include the below listed header file in each source file using a component of the Library interface. Also, edit the include file search path to locate the header file in the below listed directory. At link time the Library's shared object file is linked via the linker command line. This can be done by naming the `.so` file explicitly or by adding the below linker command line argument. At run time the library is found in the directory `/usr/lib/`. (The shared object file is automatically copied to `/usr/lib/` when it is built.)

Description	File	Location	Linker Argument
Header File	plxeeprom_api.h	.../include/	
Shared Object Library	libplxeeprom_api.so	.../lib/ /usr/lib/	-lplxeeprom_api

### 4.4. Macros

The API Library and driver interfaces include the following macros, which are defined in `plxeeprom.h`.

#### 4.4.1. EEPROM Register Identifier

The EEPROM content is accessed via the register access IOCTL services. The API includes no predefined register style EEPROM macros. Instead, applications generate the identifier values as required using the below macro.

**NOTE:** The EEPROM content is accessible as 8-bits, 16-bits or 32-bits. The driver requires that all offsets be aligned according to the associated size. Access for 8-bit values can be byte aligned. Access for 16-bit values must be word aligned (2-bytes). Access for 32-bit values must be long word aligned (4-bytes).

##### Definition

```
reg = PLX_EEPROM_REG_ENCODE(size, offset);
```

Argument	Description
size	This is the register size. The entire register content is accessible as bytes ( <code>size = 1</code> ), words ( <code>size = 2</code> ) and long word ( <code>size = 4</code> ).
offset	This is the offset of the register within the EEPROM. The value must be aligned according to the size and must not extend past the end of the EEPROM.

Return Value	Description
> 0	The result is a value used by the driver to access the desired EEPROM content.

#### 4.4.2. IOCTL Services

The IOCTL macros are documented in section 4.7 (page 23).

#### 4.4.3. Registers

The following gives the complete set of plxeprom registers.

##### 4.4.3.1. EEPROM Registers

While the EEPROM is accessed via the driver register access IOCTL services, no predefined register identifiers are provided. The identifier values are generated as needed (section 4.4.1, page 18). The plxeprom device driver provides read/write access to the entire EEPROM, based on the detected size of the installed EEPROM.

##### 4.4.3.2. PCI Configuration Registers

Access to PCI registers is seldom required so none are listed here. For the complete list of PCI register identifiers refer to the header files given in the table below. The header files can be found in the `.../include/` subdirectory.

PLX Chip	Header File
PCI9056	<code>gsc_pci9056.h</code>
PCI9060	<code>gsc_pci9060es.h</code>
PCI9080	<code>gsc_pci9080</code>
PCI9656	<code>gsc_pci9656.h</code>

##### 4.4.3.3. PLX Feature Set Registers

Access to the PLX registers is seldom required so these registers are not listed here. For the complete list of the PLX register identifiers refer to the header files given in the table above. The plxeprom interface provides read-only access to the PLX registers.

#### 4.4.3.4. GSC Firmware Registers

The plxeprom interface does not provide access to GSC registers, which are accessed via BAR2.

### 4.5. Data Types

The data types used by the API Library are described with the IOCTL services with which they are used. For additional information refer to section 4.7 (page 23).

### 4.6. Functions

The interface includes the following functions. The return values reflect the completion status of the requested operation. A return value less than zero always reflects an error condition. The table below summarizes the error status values. For the I/O function, read, non-negative return values reflect the number of bytes transferred between the application and the interface. A value equal to the requested transfer size indicates complete success. Return values less than the requested transfer size indicate that the I/O timeout expired. For the other API function calls a return value of zero indicates success.

Return Value	Description
< 0	This is the value “(-errno)” (see errno.h).

#### 4.6.1. plxeprom\_close()

This function is the entry point to close a connection made via the API's open call (section 4.6.4, page 21). The interface is put in an initialized state before this call returns.

**NOTE:** This service does not perform any operation involving the General Standards firmware or its registers.

#### Prototype

```
int plxeprom_close(int fd);
```

Argument	Description
fd	This is the file descriptor obtained from the open service (section 4.6.4, page 21).

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. See error value description above.

#### Example

```
#include <stdio.h>

#include "plxeprom_dsl.h"

int plxeprom_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = plxeprom_close(fd);

    if (ret)
```

```

        printf("ERROR: plxeprom_close() returned %d\n", ret);

    errs    = ret ? 1 : 0;
    return(errs);
}

```

#### 4.6.2. plxeprom\_init()

This function is the entry point to initializing the plxeprom API Library and must be the first call into the Library. This function may be called more than once, but only the first successful call actually initializes the library. Subsequent calls perform no operation at all. All other API calls return a failure status when the API Library is not initialized.

##### Prototype

```
int plxeprom_init(void);
```

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. See error value description above.

##### Example

```

#include <stdio.h>

#include "plxeprom_dsl.h"

int plxeprom_init_dsl(void)
{
    int errs;
    int ret;

    ret = plxeprom_init();

    if (ret)
        printf("ERROR: plxeprom_init() returned %d\n", ret);

    errs    = ret ? 1 : 0;
    return(errs);
}

```

#### 4.6.3. plxeprom\_ioctl()

This function is the entry point to performing setup and control operations on a plxeprom accessed device. This function should only be called after a successful open of the respective device. The specific operation performed varies according to the request argument. The request argument also governs the use and interpretation of the arg argument. The set of supported options for the request argument consists of the IOCTL services supported by the driver, which are defined in section 4.7 (page 23).

**NOTE:** IOCTL operations are not supported for an open on device index -1.

##### Prototype

```
int plxeprom_ioctl(int fd, int request, void* arg);
```

Argument	Description
fd	This is the file descriptor obtained from the open service (section 4.6.4, page 21).
request	This specifies the desired operation to be performed (section 4.7, page 23).
arg	This is specific to the IOCTL operation specified by the request argument.

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. See error value description above.

#### Example

```
#include <stdio.h>

#include "plxeprom_dsl.h"

int plxeprom_ioctl_dsl(int fd, int request, void* arg)
{
    int errs;
    int ret;

    ret = plxeprom_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: plxeprom_ioctl() returned %d\n", ret);

    errs = ret ? 1 : 0;
    return(errs);
}
```

#### 4.6.4. plxeprom\_open()

This function is the entry point to open a connection to a plxeprom accessed board. Before returning, the initialize IOCTL service is called to reset all software settings to their defaults.

#### Prototype

```
int plxeprom_open(int device, int share, int* fd);
```

Argument	Description						
device	This is the zero-based index of the device to access. †						
share	Open the device in Shared Access Mode? If non-zero the device is opened in Shared Access Mode (see below). If zero the device is opened in Exclusive Access Mode (see below).						
fd	The device handle is returned here. The pointer cannot be NULL. Values returned are as follows. <table border="1"> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>&gt;= 0</td><td>This is the handle to use to access the device in subsequent calls.</td></tr> <tr> <td>-1</td><td>There was an error. The device is not accessible.</td></tr> </table>	Value	Description	>= 0	This is the handle to use to access the device in subsequent calls.	-1	There was an error. The device is not accessible.
Value	Description						
>= 0	This is the handle to use to access the device in subsequent calls.						
-1	There was an error. The device is not accessible.						

† The index value -1 can also be given to acquire driver information (section 2.2, page 11).

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. See error value description above.

**Example**

```

#include <stdio.h>

#include "plxeprom_dsl.h"

int plxeprom_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = plxeprom_open(device, share, fd);

    if (ret)
        printf("ERROR: plxeprom_open() returned %d\n", ret);

    errs = ret ? 1 : 0;
    return(errs);
}

```

**4.6.4.1. Access Modes**

The value of the `share` argument determines the device access mode, as follows.

**Shared Access Mode:**

Shared Access Mode allows multiple applications to access the same device simultaneously. In this mode, the first successful open request returns with the interface in an initialized state. Subsequent successful Shared Access Mode open requests do not affect the state of the device. Once opened in Shared Access Mode, the device access remains in this mode until all Shared Access Mode accesses release the device with a close request.

**Exclusive Access Mode:**

Exclusive Access Mode allows a single application to acquire exclusive access to a device. In this mode, a successful open request returns with the interface in an initialized state. While open in this mode all subsequent open requests will fail regardless of the access mode requested. Once opened in Exclusive Access Mode, the device access remains in this mode until released by the application with a close request.

**4.6.5. plxeprom\_read()**

This function is the entry point to reading data from an open connection on device index `-1`. The function reads up to `bytes` bytes. The return value is the number of bytes actually read. The source for the data returned is the `/proc/plxeprom` text file (section 2.2, page 11).

**Prototype**

```
int plxeprom_read(int fd, void* dst, size_t bytes);
```

Argument	Description
<code>fd</code>	This is the file descriptor obtained from the open service (section 4.6.4, page 21).
<code>dst</code>	The data read is put here.
<code>bytes</code>	This is the desired number of bytes to read.

Return Value	Description
0 to bytes	The operation succeeded.
< 0	An error occurred. See error value description above.

#### Example

```
#include <stdio.h>

#include "plxeprom_dsl.h"

int plxeprom_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = plxeprom_read(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: plxeprom_read() returned %d\n", ret);

    if (qty)
        qty[0] = (ret < 0) ? 0 : (size_t) ret;

    errs = (ret < 0) ? 1 : 0;
    return(errs);
}
```

## 4.7. IOCTL Services

The plxeprom API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `plxeprom_ioctl()` function arguments.

### 4.7.1. PLXEEPROM\_IOCTL\_INITIALIZE

This service initializes the software interface for the device being accessed. This service does not access or initialize any hardware.

#### Usage

Argument	Description
request	PLXEEPROM_IOCTL_INITIALIZE
arg	Not Used

### 4.7.2. PLXEEPROM\_IOCTL\_QUERY

This service queries the driver for various pieces of information about the board and the driver.

#### Usage

Argument	Description
request	PLXEEPROM_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description															
PLXEEPROM_QUERY_DEVICE_TYPE	<p>This identifies the PLX device type. The value returned is according to the following table.</p> <table><tr><th>PLX Device</th><th>Device Type</th></tr><tr><td>PCI9056</td><td>0x9056</td></tr><tr><td>PCI9060</td><td>0x9060</td></tr><tr><td>PCI9080</td><td>0x9080</td></tr><tr><td>PCI9656</td><td>6x9656</td></tr></table>	PLX Device	Device Type	PCI9056	0x9056	PCI9060	0x9060	PCI9080	0x9080	PCI9656	6x9656					
PLX Device	Device Type															
PCI9056	0x9056															
PCI9060	0x9060															
PCI9080	0x9080															
PCI9656	6x9656															
PLXEEPROM_QUERY_EEPROM_CFG	<p>This is the size (in bytes) of the EEPROM space used for PLX configuration as given in the table below. The value is zero if the EEPROM is absent.</p> <table><tr><th>PLX Device</th><th>Size</th></tr><tr><td>PCI9056</td><td>68 or 100</td></tr><tr><td>PCI9060</td><td>68 †</td></tr><tr><td>PCI9080</td><td>68 or 88</td></tr><tr><td>PCI9656</td><td>68 or 100</td></tr></table> <p>† The size is assumed.</p>	PLX Device	Size	PCI9056	68 or 100	PCI9060	68 †	PCI9080	68 or 88	PCI9656	68 or 100					
PLX Device	Size															
PCI9056	68 or 100															
PCI9060	68 †															
PCI9080	68 or 88															
PCI9656	68 or 100															
PLXEEPROM_QUERY_EEPROM_PROG	<p>This is one if the driver is able to modify the EEPROM and zero if it can't. Support for programming is according to the table below.</p> <table><tr><th>PLX Device</th><th>Return Value</th><th>Supported</th></tr><tr><td>PCI9056</td><td>1</td><td>Yes</td></tr><tr><td>PCI9060</td><td>0</td><td>No †</td></tr><tr><td>PCI9080</td><td>0</td><td>No †</td></tr><tr><td>PCI9656</td><td>1</td><td>Yes</td></tr></table> <p>† Support is not implemented in the driver.</p>	PLX Device	Return Value	Supported	PCI9056	1	Yes	PCI9060	0	No †	PCI9080	0	No †	PCI9656	1	Yes
PLX Device	Return Value	Supported														
PCI9056	1	Yes														
PCI9060	0	No †														
PCI9080	0	No †														
PCI9656	1	Yes														
PLXEEPROM_QUERY_EEPROM_PROT	<p>This is the default size (in bytes) of the EEPROM space that is write protected upon startup.</p> <table><tr><th>PLX Device</th><th>Size</th></tr><tr><td>PCI9056</td><td>192 ‡</td></tr><tr><td>PCI9060</td><td>0 †</td></tr><tr><td>PCI9080</td><td>0 †</td></tr><tr><td>PCI9656</td><td>192 ‡</td></tr></table> <p>† There is no documented write protection feature. ‡ This is bypassed by the driver when making changes.</p>	PLX Device	Size	PCI9056	192 ‡	PCI9060	0 †	PCI9080	0 †	PCI9656	192 ‡					
PLX Device	Size															
PCI9056	192 ‡															
PCI9060	0 †															
PCI9080	0 †															
PCI9656	192 ‡															
PLXEEPROM_QUERY_EEPROM_SIZE	<p>This is the assumed size (in bytes) of the installed EEPROM. The size may actually be larger, but that is not verified by the driver.</p> <table><tr><th>PLX Device</th><th>Size</th></tr><tr><td>PCI9056</td><td>256</td></tr><tr><td>PCI9060</td><td>128</td></tr><tr><td>PCI9080</td><td>128</td></tr><tr><td>PCI9656</td><td>256</td></tr></table>	PLX Device	Size	PCI9056	256	PCI9060	128	PCI9080	128	PCI9656	256					
PLX Device	Size															
PCI9056	256															
PCI9060	128															
PCI9080	128															
PCI9656	256															
PLXEEPROM_QUERY_EEPROM_STAT	<p>The reports the status of the board's PLX EEPROM. See the EEPROM Status table below.</p>															

Valid return values are as indicated in the above table and as given in the below table.

Value	Description
PLXEEPROM_IOCTL_QUERY_ERROR	Either there was a processing error or the query option is unrecognized.



The following table explains the options returned for EEPROM Status queries.

Status	Description
PLXEEPROM EEPROM STAT NOSUP	Access to the EEPROM is not supported.
PLXEEPROM EEPROM STAT UNKNOWN	The status of the EEPROM is unknown.
PLXEEPROM EEPROM STAT ABSENT	The EEPROM is not installed.
PLXEEPROM EEPROM STAT EMPTY	The EEPROM is installed, but is not programmed.
PLXEEPROM EEPROM STAT PROG	The EEPROM is present and programmed.

#### 4.7.3. PLXEEPROM\_IOCTL\_REG\_MOD

This service performs a read-modify-write of a device register. This pertains only to the EEPROM content. The EEPROM content can be modified via this service using the EEPROM register definition values created per section 4.4.1 (page 18). The PCI and PLX registers are read-only. The GSC registers are inaccessible.

##### Register Accessibility

PLX Chip	EEPROM	PCI Registers	PLX Registers	GSC Registers
PCI9056	Yes	No, Read-Only	No, Read-Only	No, Not Accessible
PCI9060	No †	No, Read-Only	No, Read-Only	No, Not Accessible
PCI9080	No †	No, Read-Only	No, Read-Only	No, Not Accessible
PCI9656	Yes	No, Read-Only	No, Read-Only	No, Not Accessible

† Support not implemented.

##### Usage

Argument	Description
request	PLXEEPROM_IOCTL_REG_MOD
arg	gsc_reg_t*

##### Definition

```
typedef struct
{
    u32 reg;
    u32 value;
    u32 mask;
} gsc_reg_t;
```

Fields	Description
reg	This is set to the identifier for the register to access (section 4.4.1, page 18).
value	This contains the value for the register bits to modify.
mask	This specifies the set of bits to modify. If a bit here is set, then the respective register bits is modified. If a bit here is zero, then the respective register bit is unmodified.

#### 4.7.4. PLXEEPROM\_IOCTL\_REG\_READ

This service reads the value of a device register. This pertains to the EEPROM content as well as the PCI and PLX registers. The EEPROM content can be read via this service using the EEPROM register definition values created per section 4.4.1 (page 18). The PCI and PLX registers can be accessed using existing register identifiers defined in the respective header files. GSC registers are not accessible.

## Register Accessibility

PLX Chip	EEPROM	PCI Registers	PLX Registers	GSC Registers
PCI9056	Yes	Yes	Yes	No, Not Accessible
PCI9060	No †	Yes	Yes	No, Not Accessible
PCI9080	No †	Yes	Yes	No, Not Accessible
PCI9656	Yes	Yes	Yes	No, Not Accessible

† Support not implemented.

## Usage

Argument	Description
request	PLXEEPROM_IOCTL_REG_READ
arg	gsc_reg_t*

## Definition

```
typedef struct
{
    u32 reg;
    u32 value;
    u32 mask;
} gsc_reg_t;
```

Fields	Description
reg	This is set to the identifier for the register to access (section 4.4.1, page 18).
value	This is the value read from the specified register.
mask	This is ignored for read request.

### 4.7.5. PLXEEPROM\_IOCTL\_REG\_WRITE

This service writes a value to a device register. This pertains only to the EEPROM content. The EEPROM content can be written via this service using the EEPROM register definition values created per section 4.4.1 (page 18). The PCI and PLX registers are read-only. The GSC registers are inaccessible.

## Register Accessibility

PLX Chip	EEPROM	PCI Registers	PLX Registers	GSC Registers
PCI9056	Yes	No, Read-Only	No, Read-Only	No, Not Accessible
PCI9060	No †	No, Read-Only	No, Read-Only	No, Not Accessible
PCI9080	No †	No, Read-Only	No, Read-Only	No, Not Accessible
PCI9656	Yes	No, Read-Only	No, Read-Only	No, Not Accessible

† Support not implemented.

## Usage

Argument	Description
request	PLXEEPROM_IOCTL_REG_WRITE
arg	gsc_reg_t*

## Definition

```
typedef struct
{
```

```

    u32 reg;
    u32 value;
    u32 mask;
} gsc_reg_t;

```

Fields	Description
reg	This is set to the identifier for the register to access (section 4.4.1, page 18).
value	This is the value to write to the specified register.
mask	This is ignored for write request.

## 5. The Driver

**NOTE:** Contact General Standards Corporation if additional driver functionality is required.

### 5.1. Files

The device driver files are summarized in the table below.

Description	Files	Location
Source Files	*.c, *.h ...	.../driver/
Header File	plxeprom.h	
Driver File	plxeprom.ko † plxeprom.o ‡	

† This is for kernel versions 2.6 and later.

‡ This is for kernel versions 2.4 are earlier.

### 5.2. Build

**NOTE:** Building the driver requires installation of the kernel headers and possibly other packages.

The device driver is built via the Overall Make Script (section 2.7, page 13), but can be built separately following the below steps.

1. Change to the directory where the driver and its sources are installed (.../driver/).
2. Remove existing build targets by issuing the below command.

```
make clean
```

3. Build the driver by issuing the below command.

```
make
```

**NOTE:** Due to the differences between the many Linux distributions some build errors may occur. These errors may include system header location differences, which should be easily corrected.

### 5.3. Startup

**NOTE:** The driver will have to be built before being used as it is provided in source form only.

The startup script used in this procedure is designed to load the device driver and create fresh device nodes. This is accomplished by unloading the current driver, if loaded, and then loading the accompanying driver executable. In addition, the script deletes and recreates the device nodes. This is done to ensure that the device nodes in use have the same major number as assigned dynamically to the driver by the kernel, and so that the number of device nodes corresponds to the number of boards identified by the driver.

#### 5.3.1. Manual Driver Startup Procedures

Start the driver manually by following the below listed steps.

**NOTE:** The following steps may require elevated privileges.

1. Change to the directory where the driver sources are installed (.../driver/).
2. Install the driver module and create the device nodes by executing the below command. If any errors are encountered then an appropriate error message will be displayed.

```
./start
```

**NOTE:** This script must be executed each time the host is booted.

**NOTE:** The plxeprom device node major number is assigned dynamically by the kernel. The minor numbers and the device node suffix numbers are index numbers beginning with zero, and increase by one for each additional board installed.

3. Verify that the device driver module has been loaded by issuing the below command and examining the output. The module name `plxeprom` should be included in the output.

```
lsmod
```

4. Verify that the device nodes have been created by issuing the below command and examining the output. The output should include one node for each installed board.

```
ls -l /dev/plxeprom.*
```

### 5.3.2. Automatic Driver Startup Procedures

Start the driver automatically with each system reboot by following the below listed steps.

1. Locate and edit the system startup script `rc.local`, which should be in the `/etc/rc.d/` directory. Modify the file by adding the below line so that it is executed with every reboot. The example is based on the driver being installed in `/usr/src/linux/drivers/`, though it may have been installed elsewhere.

```
/usr/src/linux/drivers/plxeprom/driver/start
```

**NOTE:** For `systemd` installations the file `rc.local` may be located under the `/etc/` directory rather than under `/etc/rc.d/`.

2. Load the driver and create the required device nodes by rebooting the system.
3. Verify that the driver is loaded and that the device nodes have been created. Do this by following the verification steps given in the manual startup procedures.

#### 5.3.2.1. File `rc.local` Not Present

Some distributions may not install a default version of `rc.local`. Some may not even create the directory `/etc/rc.d/`. If the directory is not present, then it may be created. The directory must be created with the owner and group set to `root`. The directory permissions must be set to `rwxr-xr-x`. If the file `/etc/rc.d/rc.local` is not present, then it too may be created. The file must also be created with the owner and group set to `root`. Additionally, the file permissions must also be set to `rwxr-xr-x`. After the directory and file are created as described, reboot to verify boot time loading of the driver. Here is an example of a default version of `rc.local`.

```
#!/bin/bash

# Add your local content here.
```

### 5.3.2.2. Default `rc.local` File Permissions

The `rc.local` script may fail to run at boot time because some distributions install a default version of the file without execute permissions. Without execute permissions, boot time invocation of the script fails, which inhibits boot time loading of the driver. If this is the case, then change the file permissions to `rxwxr-xr-x`. After the file permissions are adjusted as described, reboot to verify boot time loading of the driver.

### 5.3.2.3. `systemd` Installations

With the advent of the `systemd` startup implementation, `rc.local` may be accessed via a `systemd` startup service. The service name may be `rc-local`, `rc-local.service` or something similar. This service may or may not be enabled by default. If the service is disabled, then the script will not execute, which prevents boot time loading of the driver. The service can be enabled with the below command line. After the service is enabled, reboot to verify boot time loading of the driver.

```
systemctl enable rc-local
```

**NOTE:** For `systemd` installations the file `rc.local` may be located under the `/etc/` directory rather than under `/etc/rc.d/`.

### 5.3.2.4. `systemd` and `rc.local` Timing

If the above steps have been performed but the driver still does not start then examine the `dmesg` output for driver messages. If the output shows that the driver starts and immediately stops, then the problem may be timing. That is, since `systemd` doesn't serialize startup initialization as done in the past, driver loading may fail if required services have not completed their own initialization. If this is the problem, then it may be corrected simply by inserting a delay in `rc.local` prior to it calling the driver's start script (i.e., sleep for one or more seconds).

### 5.3.2.5. SELinux Implications

If not disabled, then SELinux may prevent boot time loading of the driver. If this is the case, then it can be verified and corrected using SELinux related tools and utilities. First, install the necessary software using the below command. (As necessary, replace the `yum` command line with that which is available for your distribution.)

```
yum install setroubleshoot setools
```

Next, run the below command to determine if SELinux is preventing the driver from loading at boot time.

```
sealert -a /var/log/audit/audit.log
```

If SELinux is preventing the driver from loading, then the output from the above command should include a reference to the driver's start script, the `insmod` command that loads the driver or the name of the driver executable. If so, then the output should also indicate the commands necessary to resolve the issue. The following is an example of the instructions given when the culprit is `insmod`, which is the start script command that loads the driver. After running these commands reboot the system to verify boot time loading of the driver.

```
ausearch -c 'insmod' --raw | audit2allow -M my-insmod
semodule -X 300 -i my-insmod.pp
```

## 5.4. Verification

Follow the below steps to verify that the driver has been properly installed and started.

1. Verify that the file `/proc/plxeprom` is present. If the file is present then the driver is loaded and running. Verify the file's presence by viewing its content with the below command.

```
cat /proc/plxeprom
```

## 5.5. Version

The driver version number can be obtained in a variety of ways. It is reported by the driver both when the driver is loaded and when it is unloaded (depending on kernel configuration options, this may be visible only in places such as `/var/log/messages`). It is reported in the text file `/proc/plxeprom` while the driver is loaded and running. The version number is also given in the file `release.txt` in the root install directory.

## 5.6. Shutdown

Shutdown the driver following the below listed steps.

**NOTE:** The following steps may require elevated privileges.

1. If the driver is currently loaded then issue the below command to unload the driver.

```
rmmod plxeprom
```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `plxeprom` should not be in the listed output.

```
lsmod
```

## 6. Document Source Code Examples

The source code examples included in this document are built into a statically linkable library usable with console applications. The purpose of these files is to verify that the documentation samples compile and to provide a library of working sample code to assist in a user's learning curve and application development effort.

### 6.1. Files

The library files are summarized in the table below.

Description	Files	Location
Source Files	*.c, *.h ...	.../docsrc/
Header File	plxeprom_dsl.h	.../include/
Library File	plxeprom_dsl.a	.../lib/

### 6.2. Build

The library is built via the Overall Make Script (section 2.7, page 13), but can be built separately following the below steps.

1. Change to the directory where the documentation sources are installed (.../docsrc/).
2. Remove existing build targets by issuing the below command.

```
make clean
```

3. Compile the sample files and build the library by issuing the below command.

```
make
```

4. Rebuild the Main Library (section 3.2.1, page 15).

### 6.3. Library Use

The library is used both at application compile time and at application link time. At compile time include the above listed header file in each source file using a component of the library interface. At link time include the above listed static library file with the objects being linked with the application.



## 7. Utilities Source Code

The API Library installation includes a body of utility source code designed to aid in the understanding and use of the interface calls and IOCTL services. Utility sources are also included for device independent and common, general-purpose services. Most of the utilities are implemented as visual wrappers around the corresponding services to facilitate structured console output for the sample applications. The utility sources are compiled and linked into static libraries to simplify their use. An additional purpose of these utility services is to provide a library of working sample code to assist in a user's learning curve and application development effort.

For each API function there is a corresponding utility source file with a corresponding utility service. As an example, for the API function `plxeprom_open()` there is the utility file `open.c` containing the utility function `plxeprom_open_util()`. The naming pattern is as follows: API function `plxeprom_xxxx()`, utility file name `xxxx.c`, utility function `plxeprom_xxxx_util()`. Additionally, for each IOCTL code there is a corresponding utility source file with a corresponding utility service. As an example, for IOCTL code `PLXEEPROM_IOCTL_QUERY` there is the utility file `query.c` containing the utility function `plxeprom_query()`. The naming pattern is as follows: IOCTL code `PLXEEPROM_IOCTL_XXXX`, utility file name `xxxx.c`, utility function `plxeprom_xxxx()`.

### 7.1. Files

The utility files are summarized in the table below.

Description	Files	Location
Source Files	*.c, *.h ...	.../utils/
Header File	plxeprom_utils.h	.../include/
Library Files	plxeprom_utils.a gsc_utils.a os_utils.a plx_utils.a	.../lib/

### 7.2. Build

The libraries are built via the Overall Make Script (section 2.7, page 13), but can be built separately following the below steps.

1. Change to the directory where the utility sources are installed (.../utils/).
2. Remove existing build targets by issuing the below command.

```
make clean
```

3. Compile the sample files and build the library by issuing the below command.

```
make
```

4. Rebuild the Main Library (section 3.2, page 15).

### 7.3. Library Use

The library is used both at application compile time and at application link time. At compile time include the above listed header file in each source file using a component of the library interface. At link time include the above listed static library file with the objects being linked with the application.

## 8. Operating Information

This section explains some basic operational procedures for using the plxeprom driver and the plxeprom software utility. This is in no way intended to be a comprehensive guide. This is simply to address a very few issues relating to their use.

### 8.1. Debugging Aids

The driver package includes the following items useful for development and/or debugging aids.

#### 8.1.1. Device Identification

When communicating with technical support complete device identification is virtually always necessary. The *id* example application is provided for this specific purpose. This is a text only console application. The output can be piped to a file, which can then be emailed to GSC technical support when requested. Locate the application as follows.

Description	File	Location
Application	id	.../id/

### 8.2. PLX EEPROM Support

The plxeprom driver supports only a limited set of PLX bridge chips. Plus, the level of support for access to device resources varies according to the PLX chip in use and the resource being accessed. Both chip support and resource access are identified in the table below.

PLX Chip	Supported	Register Access			
		EEPROM	PCI	PLX	GSC
PCI9056	Yes	Read/Write	Read-Only	Read-Only	Not Accessible ‡
PCI9060	Yes	None †	Read-Only	Read-Only	Not Accessible ‡
PCI9080	Yes	None †	Read-Only	Read-Only	Not Accessible ‡
PCI9656	Yes	Read/Write	Read-Only	Read-Only	Not Accessible ‡

† The driver does not presently implement access to this chip's configuration EEPROM.

‡ The driver does not provide access to these registers.

### 8.3. PLX EEPROM Organization

The EEPROMs used by the supported PLX chips all organize data in a similar fashion, though not in the same size segments. Some EEPROM characteristics are assumed by the driver and some are determined at run time. These characteristics are reported to and used by applications, including the plxeprom application.

PLX Chip	EEPROM Size	Configuration Size	Unused Size
PCI9056	256 Bytes †	100 Bytes ‡	156 Bytes *
PCI9060	128 Bytes †	68 Bytes ‡	60 Bytes *
PCI9080	128 Bytes †	88 Bytes ‡	40 Bytes *
PCI9656	256 Bytes †	100 Bytes ‡	156 Bytes *

† This is the smallest assumed size, though it might be larger.

‡ The configuration size is determined at run time and may be smaller.

\* This is the assumed size based on figures at left. The size may be larger only if the EEPROM size is larger.

## 8.4. plxeprom Software Utility

Access to, and modification of, the PLX configuration EEPROM is the primary purpose behind the plxeprom driver package. Within that package is the plxeprom software utility, which is the provided means for making changes to the PLX configuration portion of the EEPROM. To use this utility, open a command shell and change to the directory where the sample application is located (.../plxeprom/).

**NOTE:** The PLX configuration EEPROM may have content besides that used to configure the PLX bridge chip. An example of such added content may be the optional Vital Product Data (VPD). The plxeprom utility does not provide tailored access to such content.

### 8.4.1. Identify What Is Installed

**WARNING:** The plxeprom device driver recognizes all PLX bridge chips from the above table based on having a PLX Vendor ID of 10B5, even if it is for a device not produced by General Standards. Be very careful not to change configuration data for non-General Standards products. Doing so could have no effect, make the device useless or cause damage to the device or other equipment.

Before using plxeprom one should first verify the set of devices that are installed. The first approach is with the below command line. Examine the output from the command, ignoring entries not listed in the above table. This should list all devices that will be recognized by the plxeprom device driver. If the list appears to correspond to the set of GSC boards known to be installed, then that is a good sign. The concern would be if the set of listed devices does not correspond. If they don't correspond, then it simply means the plxeprom driver may detect non-GSC products. If so, then extra care should be exercised to make sure the driver is not used to modify the EEPROM from those other products.

```
lspci | grep -i plx
```

Next, with the plxeprom software having been installed and the driver started, use the installed content to take a look at what the driver recognizes. To do this, look at the text file created by the driver in the /proc/ directory (section 2.2, page 11). Execute the below command and examine the output. Ideally, each “models” item listed will reflect the base model number of the GSC boards known to be installed. (In some cases, the model numbers may only be similar.) Any model given as “other” means the plxeprom driver recognized a device that may be a non-GSC product. Exercise care in accessing such devices. If it is a non-GSC product, then do not attempt to change its PLX configuration EEPROM.

```
cat /proc/plxeprom
```

**NOTE:** Model numbers listed in the /proc/plxeprom text file are based on data hardcoded into the driver. The driver will periodically be updated, but at any given time the hardcoded data may be out of date. Devices in the hardcoded list will be named. In some cases, the model number given will be similar, but not necessarily correct. Devices named as “GSC” are not in the hardcoded list, but are recognized as GSC products. Model numbers will otherwise be listed as “other” and may be non-GSC products.

For final verification, or clarification, of what is detected by the driver, one can use the id sample application. For this, change to the directory where the application is installed (.../id/) and issue the below command for each device detected by the driver. In the command, replace the “X” with the zero-based index of each device detected by the driver. The board identification given in the output should correspond with model numbers given in the /proc/ test file.

```
./id X
```

### 8.4.2. Backup The EEPROM Configuration Image

Before attempting any changes to a device's EEPROM content, be sure to back up the EEPROM's configuration image to disk. This can be done by issuing the below command. The application will save the configuration image to a `.eep` file. The file name will include the PLX chip model number reported from the PLX hard coded Device ID register, the term `"cfg"` followed by the current date and time.

```
./plxeprom -cfgs
```

The `"cfg"` refers to the PLX configuration portion of the EEPROM. The `"s"` specifies that the data is to be saved to disk.

An example output file name is as follows.

```
PCI9056_cfg_20250811_135423.eep
```

### 8.4.3. In Case of An Error, Restore the EEPROM Configuration Image

If a problem arises while applying changes to the EEPROM's PLX configuration image, restore the original image. This can be done by changing to the application's directory (`.../plxeprom/`) and issuing the below command. In this case, the file being restored is the one created from the above example.

```
./plxeprom -cfgu -cfgf=16A016_PCI9056_cfg_20250811_135423.eep
```

The `"cfg"` refers to the PLX configuration portion of the EEPROM. The `"u"` specifies that the data is to be updated from a disk image. The `"f"` refers to the file name from which to restore the image. Both command line arguments are required to update the current configuration data to content from a file.

### 8.4.4. Device Identity

With the command line argument given below, the application will report the device identity currently in use, the hard coded identity from the PLX chip and the identity recorded in the EEPROM.

Argument	Description
<code>-id</code>	Display the device identify from all available sources.

### 8.4.5. Vendor ID

The `plxeprom` application can be used to access and change the configured Vendor ID (VID) using the arguments given in the below table. When a change is specified the application writes the specified value to the EEPROM, reads the current value recorded in the EEPROM, then reports and identifies the current value from the EEPROM.

**NOTE:** The `plxeprom` device driver does not, by default, recognize Vendor ID values other than the PLX value of 10B5 and the GSC value of 1C6E. To recognize any other value, the identification table at the top of the driver source file `device.c` must be expanded.

Argument	Description
<code>-vid</code>	Display the VID from all available sources.
<code>-vidgsc</code>	Change the EEPROM recorded VID to the GSC specific value 0x1C6E. This is rarely used. †‡
<code>-vidplx</code>	Change the EEPROM recorded VID to the PLX specific value 0x10B5. This is the expected value.

† This `plxeprom` driver automatically recognizes any device with this Vendor ID as a General Standards product.

‡ Very few drivers recognize 0x1C6E as belonging the General Standards.

#### 8.4.6. Device ID

The plxeprom application can be used to access and change the configured Device ID (DID) using the arguments given in the below table. When a change is specified the application writes the specified value to the EEPROM, reads the current value recorded in the EEPROM, then reports and identifies the current value from the EEPROM.

**WARNING:** If the Device ID is set to a value not reflecting the actual PLX chip installed, software access to the device and the EEPROM may require an alternate method once the system is rebooted.

Argument	Description
-did	Display the DID from all available sources.
-did9056	Change the EEPROM recorded DID to the value for the PCI9056.
-did9060	Change the EEPROM recorded DID to the value for the PCI9060.
-did9080	Change the EEPROM recorded DID to the value for the PCI9080.
-did9656	Change the EEPROM recorded DID to the value for the PCI9656.

#### 8.4.7. Class Code

The plxeprom application can be used to access and change the configured Class Code (CC) using the arguments given in the below table. When a change is specified the application writes the specified value to the EEPROM, reads the current value recorded in the EEPROM, then reports the current value from the EEPROM.

Argument	Description
-cc	Display the CC from all available sources.
-cc0xxxxxxx	Change the EEPROM recorded Class Code to the value 0xxxxxxx.

#### 8.4.8. BAR1 Configuration

The plxeprom application can be used to access and change the BAR1 configuration using the arguments given in the below table. When a change is specified the application writes the specified value to the EEPROM, reads the current value recorded in the EEPROM, then reports the current configuration from the EEPROM.

Argument	Description
-bar1	Display the BAR1 configuration from all available sources.
-bar1d	Change the EEPROM recorded BAR1 configuration to <i>disabled</i> .
-bar1e	Change the EEPROM recorded BAR1 configuration to <i>enabled</i> .

#### 8.4.9. BAR2 Configuration

The plxeprom application can be used to access and change the BAR2 configuration using the arguments given in the below table. When a change is specified the application writes the specified value to the EEPROM, reads the current value recorded in the EEPROM, then reports the current configuration from the EEPROM.

Argument	Description
-bar2	Display the BAR2 configuration from all available sources.
-bar2i	Change the EEPROM recorded BAR2 configuration to <i>I/O mapped</i> .
-bar2m	Change the EEPROM recorded BAR2 configuration to <i>memory mapped</i> .

#### 8.4.10. Load An Existing .EEP File

General Standards has pregenerated .eep files for some boards and certain common configurations. If such a file is provided, it can be used to program the EEPROM on a corresponding device. This can be done with the below command, where the text "pmc66-16a016-mem.eep" refers to the name of the .eep file provided.

```
./plxeprom -cfgu -cfgf=pmc66-16a016-mem.eep
```

The “`cfg`” refers to the PLX configuration portion of the EEPROM. The “`u`” specifies that the data is to be updated from a disk image. The “`f`” refers to the file name whose content is to be programmed into the board’s EEPROM.

## 9. Sample Applications

The driver archive includes a variety of sample and test applications. While they are provided without support and without any external documentation, any problems reported will be addressed as time permits. The applications are command line based and produce text output for display on a console. All of the applications are built via the Overall Make Script (section 2.7, page 13), but each may be built individually by changing to its respective directory and issuing the commands “make clean” and “make all”. The initial output from each application includes information on its supported command line arguments. The following gives a brief overview of each application.

### 9.1. id - Identify Board - .../id/

This application reports detailed board identification information. This can be used with tech support to help identify as much technical information about the board as possible from software.

### 9.2. plxeprom – PLX EEPROM Access - .../plxeprom/

This application provides access to a device’s PLX configuration EEPROM. For detailed information refer to section 8.4 (page 35).

### 9.3. regs - Register Access - .../regs/

This application provides menu based interactive access to the board’s registers, and reports other pertinent information to the console.

## Document History

Revision	Description
August 20, 2025	Initial release, version 1.2.116.52.0. Updated the kernel support table. Added section on environment variables. Updated the information for the open and close calls. Minor editorial changes. Removed references to the PLX PEX8111 and PEX8112 as they are not supported. Removed the BAR Qty query option. Added the plxeprom application.
January 28, 2021	Initial release, version 1.1.93.35.0. Updated the kernel support table. Minor editorial changes. Added a licensing subsection.
March 28, 2019	Initial release, version 1.0.83.27.0.