

# Canadian Light Source

## General ASCII Serial Line Driver Software for EPICS

### Design Note

Technical Document 7.2.39.15 Rev. 0

Date

Signature

Date

**Original on File – Signed by:**

Prepared by: \_\_\_\_\_  
Ian Stavness

Reviewed by: \_\_\_\_\_  
Tony Wilson

Reviewed by: \_\_\_\_\_  
Glen Wright

Approved by: \_\_\_\_\_  
Elder Matias

**Copyright 2001, Canadian Light Source Inc.** This document is the property of Canadian Light Source Inc. (CLS). No exploitation or transfer of any information contained herein is permitted in the absence of an agreement with CLS, and neither the document nor any such information may be released without the written consent of CLS.

Canadian Light Source  
107 North Road  
University of Saskatchewan  
Saskatoon, Saskatchewan Canada

---

### REVISION HISTORY

---

<i>Revision</i>	<i>Date</i>	<i>Description</i>	<i>Author</i>
A	2002-06-11	Original Draft Issued for Review.	Ian Stavness
0	2002-11-28	Issued for Use.	Ian Stavness

## Table of Contents

1.0	INTRODUCTION.....	1
1.1	Purpose.....	1
1.2	Scope .....	1
1.3	Abbreviations .....	1
2.0	SL DRIVER IMPLEMENTATION.....	1
2.1	Background I/O Thread .....	1
2.2	Serial Communication.....	2
2.3	SL Interface Routines .....	2
2.4	Device Specific Information .....	2
3.0	SL DRIVER API .....	3
3.1	Serial Line Initialization .....	3
3.2	ReQuest to Send Command to Serial Line .....	3
3.3	Send Command Directly to Serial Line .....	4
4.0	SUPPORTED RS232 DEVICES AND RELATED DOCUMENTS .....	5
4.1	Varian Dual Ion Pump Controller .....	5
4.2	Varian MidiVac Ion Pump Controller .....	5
5.0	APPENDIX B: I/O DIAGRAMS .....	6
5.1	Primary Thread Send Command Diagram.....	6
5.2	Background I/O Thread Diagram.....	7

## 1.0 INTRODUCTION

The Serial Line Driver software, developed by the Control and Instrumentation Group at the Canadian Light Source, is a general driver to communicate with RS-232 devices with ASCII command strings.

The driver is designed to handle the I/O processing of commands and low-level serial communication calls. Functionality that is specific to the device being controlled is to be implemented at the EPICS record level. Thus the creation of ASCII command strings and processing of ASCII response strings from the device are handled at a level above that of the Serial Line driver.

### 1.1 PURPOSE

This document is intended to outline the functionality and implementation of the Serial Line Driver Software. It will also explain the software interface and how the driver is to be used.

### 1.2 SCOPE

This document will cover how the Serial Line driver software is implemented and how the software can be used.

### 1.3 ABBREVIATIONS

SL	Serial Line
EPICS	Experimental Physics and Industrial Control System
PV	Process Variable:
I/O	Input / Output
O/S	Operating System

## 2.0 SL DRIVER IMPLEMENTATION

The Serial Line Driver software is implemented in *drvSerialLine.c*.

See section 5.0 of this document for flow diagrams of I/O for the Serial Line driver.

### 2.1 BACKGROUND I/O THREAD

A background I/O thread (*SLIOTask()*) is started for each serial line. This I/O thread is responsible for processing I/O requests from the I/O request queue that is associated with its serial line. A request to send a command with *SLCommand()* creates a new entry, with all information on the request, in the I/O Queue. Once the entry is created the background I/O Thread is signaled via an EPICS event to process the I/O Queue. Processing of an I/O entry involves sending the entry's command string to the serial line,

receiving a response if it is expected, and calling the asynchronous I/O callback function if it is available. Once all I/O Queue entries have been processed the background I/O Thread waited for another EPICS event.

**void serialLineIOTask(int serialLineNum);**

- Implementation of the background I/O Task

**void processIOQueue(int serialLineNum);**

- Processes all entries in I/O request queue

## 2.2 SERIAL COMMUNICATION

Actual communication with the device hardware is isolated to the *sendSLCommand()* routine and is made through generic SerialIO communication calls. Since the implementation of the serial communication is left to SerialIO the SL Driver maintains O/S independence.

There are currently two implementations of SerialIO. For EPICS applications running in RTEMS on an IOC, serial communication using the *sxcIO* is defined in *SerialIO\_rtems.c*. For EPICS applications running in Linux, serial communication using Linux *termios* is defined in *SerialIO\_linux.c*. Dependent on the O/S in which the driver is compiled the appropriate SerialIO library will be built.

The interface for SerialIO is as follows:

**int SerialInit(int serialLineNum);**

**int SerialRead(int serialLineNum, char \*string, unsigned int size);**

**int SerialWrite(int serialLineNum, char \*string, unsigned int size);**

**int SerialSelect(int \*, unsigned int, unsigned int);**

## 2.3 SL INTERFACE ROUTINES

An EPICS record DSET defined in device support files (*devXXXX.c*) will communicate the device hardware by using the SL interface routines.

EPICS record associated with a device connected on a serial line are responsible for initializing that serial line with the *SLInit()* routine.

To request a command string to be sent to a device on a serial line the *SLCommand()* routine is used.

To send a command string directly to a device on a serial line the *SLSend()* routine is used. This routine is designed to be used from the EPICS command prompt only.

See Section 4.0 SL Driver API for more information about SL interface routines.

## 2.4 DEVICE SPECIFIC INFORMATION

All information specific to the device being controlled by the Serial Line driver is taken out of the actual driver. The SL driver is responsible only for sending the command

string that was created by the device specific EPICS record, and passing the response string from the hardware back to the EPICS record.

No processing of the response string is done by the SL driver – the entire ASCII response string received from the hardware is returned to the EPICS record as is including the termination character.

### 3.0 SL DRIVER API

The SL Driver API routines are declared in *serialLineInterface.h*. The SL Driver API is the interface for EPICS to the SL driver.

**Note:** For all SL Interface routines the integer return value is a status value: status < 0 signifies error state.

#### 3.1 SERIAL LINE INITIALIZATION

**int SLInit(int sln, int maxResponseSize, char terminationChar, int numTerminationChar);**

Responsibilities:

- Creates a data structure (*struct lineInfo*) for all information relevant to the serial line. LineInfo contains data on max response size, response termination character, number of termination characters in response string, errors on serial line, as well as pointers to I/O Queue, I/O Thread, EPICS semaphore, and EPICS event for the I/O Thread.
- Creates I/O Queue
- Creates and starts the background I/O Thread.
- Calls IOSerial serial line initialization to setup serial line for use

Parameters:

- int sln                                   the serial line number to initialize
- int maxResponseSize                   maximum size of buffer for response string
- char terminationChar                   the character that terminates a response string from the device hardware
- int numTerminationChar               the number of times the termination character appears in a response string from the device

**Note** – if SLInit() is called and the serial line number specified has already been initialized the routine will simply return with a successful return status. Thus SLInit() can be used to check for initialization.

#### 3.2 REQUEST TO SEND COMMAND TO SERIAL LINE

**int SLCommand(int sln, char \*commandString, char \*responseString, int expectResponse, void (\*callbackFunc)(void\*), void \*userArg);**

Responsibilities:

- Sets up an I/O Queue entry with the information that is specified in the routine's parameters.
- Sends an EPICS event signal to the background I/O Thread (*serialLineIOTask()*) to wake the task.

Parameters:

- int sln the serial line number to initialize
- char \*commandString pointer to the character buffer containing the ASCII string to be sent to the serial line
- char \*responseString pointer to the character buffer to which the response ASCII character string is to be written
- int expectResponse set if a response is expected from the device after the command has been sent
- void (\*callbackFunc) (void\*) void pointer to the callback function used in asynchronous I/O – function is called after the command has been successfully sent. For synchronous I/O set to pointer to NULL.
- void \*userArg void pointer to user argument for the I/O callback function

### 3.3 SEND COMMAND DIRECTLY TO SERIAL LINE

**int SLSend(int sln, char \*commandString, char \*responseString, int expectResponse);**

Responsibilities:

- Sends the specified commandString directly to the serial line and places the response, if expected, in the responseString buffer.

Parameters:

- int sln the serial line number to initialize
- char \*commandString pointer to the character buffer containing the ASCII string to be sent to the serial line
- char \*responseString pointer to the character buffer to which the response ASCII character string is to be written
- int expectResponse set if a response is expected from the device after the command has been sent

**Note** – SLS:end() is designed to be used from the EPICS command prompt as a debugging tool. It bypasses all driver functionality of queuing and processing all send command requests.

## **4.0 SUPPORTED RS232 DEVICES AND RELATED DOCUMENTS**

Currently two devices that communicate over RS-232 are supported using the Serial Line Driver software.

### **4.1 VARIAN DUAL ION PUMP CONTROLLER**

For information on the implementation of the Varian Dual EPICS application see the following documentation:

- Varian Dual Controller Application Software for EPICS Design Note (7.2.39.16)

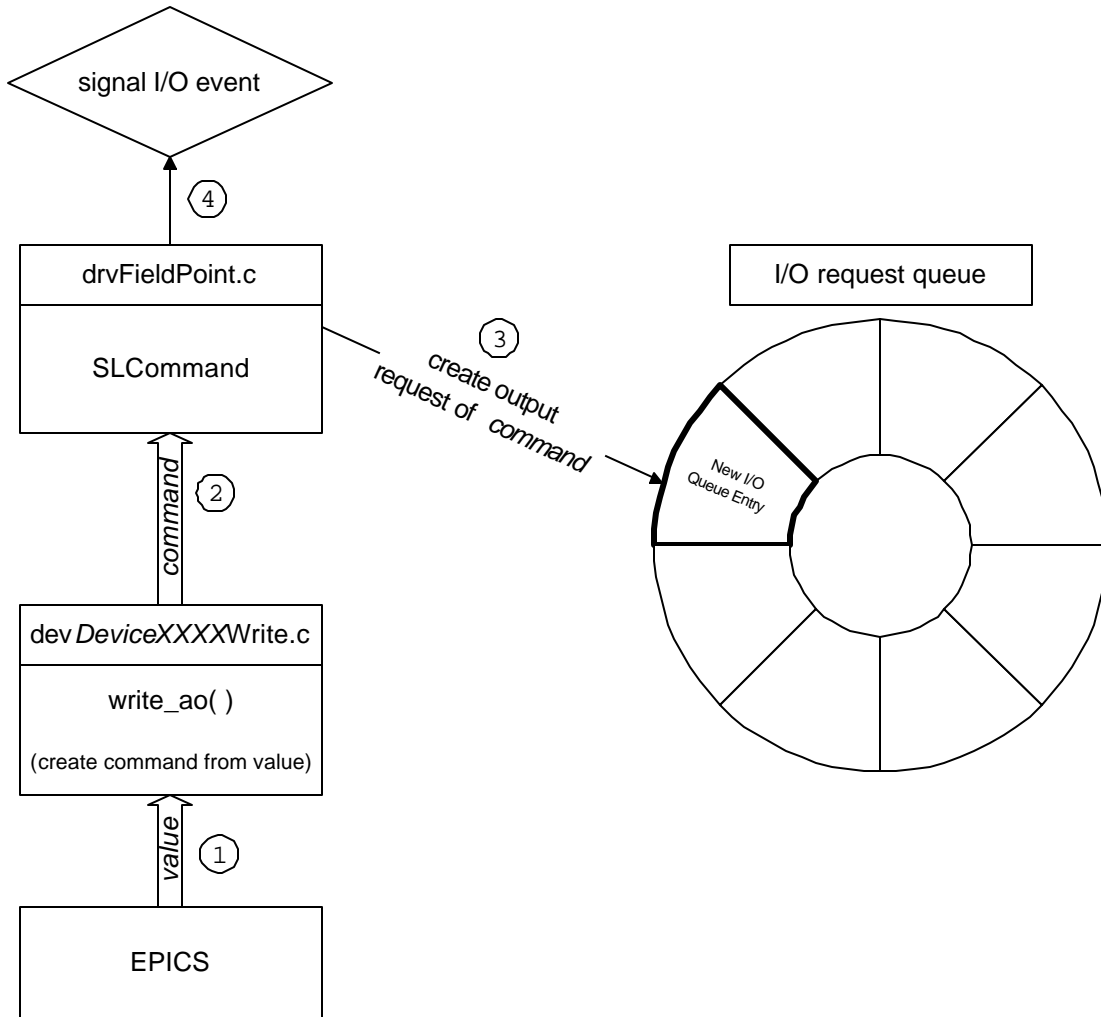
### **4.2 VARIAN MIDIVAC ION PUMP CONTROLLER**

For information on the implementation of the Varian Dual EPICS application see the following documentation:

- Varian MidiVac Controller Application Software for EPICS Design Note (7.2.39.17)

## 5.0 APPENDIX B: I/O DIAGRAMS

### 5.1 PRIMARY THREAD SEND COMMAND DIAGRAM



## 5.2 BACKGROUND I/O THREAD DIAGRAM

