Kromek Spectrometer Software Driver – Technical Manual

Version v16.3.8.23

Contents

Introduction

This document describes how to use the Windows/Linux driver for controlling the Kromek range of detectors.

Overview

This driver works by using an application level DLL. To communicate with a detector your application must interface with the DLL.

Once you are communicating with the detector data is read from the device(s) using either an asynchronous call back function or by calling a single function to return the counts across all channels for the current acquisition. Multiple detectors can be interfaced with simultaneously.

Using the Application Level Driver

To access the driver from your software you will need to use the following files:

SpectrometerDriver.dll – Found in the libs/win(32/64) folders (Windows)
SpectrometerDriver.lib - Found in the libs/win(32/64) folders (Windows)
libSpectrometerDriver.so – Found in the libs/linux(32/64) folders (Linux)

SpectrometerDriver.h– Use this header to access the library

The above Windows libraries are all built using Microsoft Visual Studio 2019. Linux library files are not provided in compiled form but can be easily built for you chosen platform using the cmake files provided.

Important Development Notes

Windows

For Windows development there are two versions of the above files provided, one for debugging and one for release. When developing your application please ensure you use the debug version of the library if you are running your application in the debug configuration and likewise the release version if you are using the release configuration. If you do not do this, you may find the USB devices are not recognised.
To run applications using the library you will need to copy the SpectrometerDriver.dll into the same directory as your executable. Again, please make sure that you are using the correct version of the dll depending on your configuration settings (debug/release).

Linux

Linux requires the use of the custom driver module that can be found in the src/ LinuxModule folder. A user guide is also provided in the folder to explain how to build and install the driver
When running your application, you may also need to set the appropriate directory to your LD_LIBRARY_PATH for the libSpectrometerDriver.so file to be found when running your application. Alternatively, the file can be copied into the correct lib directory for your Linux Distribution (such as /etc/libs).
Read and write permissions are required on any detectors you wish to use (dev/kromek*). These permissions are applied by the udev sub system and setup is normally configured by adding rules to the /etc/udev/rules.d directory. LinuxModule/Install.sh will do this automatically.

All functions are declared using the standard (__stdcall) calling type. Please ensure that any functions including any call back functions are also declared for this type.
On some linux systems, including Raspberry Pi, you may also have to add the module to the list of modules to load on system startup. Add the module 'kromekusb' to the the /etc/modules file and reboot the machine.

Spectrometer Data Format

Data is returned in one of two ways:

- Callback – A callback function can be provided receive counts as soon as they are received by the driver. The call back function also includes an estimated timestamp of when the count(s) were received. See the kr_SetDataReceivedCallback function
- Stored Data – The driver will store count data for each device. Stored data can be retrieved by calling the kr_GetAcquiredData function. Stored data can be cleared at any time by calling the kr_ClearAcquiredData function.

API Reference

Functions

kr_GetVersionInformation

Returns version information about the driver DLL.

Prototype

void kr_GetVersionInformation(int *pProduct, int *pMajor, int *pMinor, int *pBuild);

Parameters

Name	Type	Input / Output
pProduct	Int *	Output
pMajor	Int *	Output
pMinor	Int *	Output
pBuild	Int *	Output

Description

Name	Description
pProduct	A variable to receive the product version
pMajor	A variable to receive the Major version
pMinor	A variable to receive the Minor version
pBuild	A variable to receive the build version

Return Value

None

kr_SetDeviceChangedCallback

Sets a call back function for whenever a new device is plugged in or any existing device is unplugged

Prototype

void kr_SetDeviceChangedCallback(DeviceChangedCallback callbackFunc, void *pUserData);

Parameters

Name	Type	Input / Output
callbackFunc	DeviceChangedCallback (function ptr)	Input
pUserData	Void*	Input

Description

Name	Description
callbackFunc	A function to call when any new devices are connected or any devices are disconnected
pUserData	A pointer to application defined data that will automatically be passed into the call back function.

Return Value

None

Remarks

This call back function will be raised when any device changes. The id of the device will be passed into the callback as well as a BOOL value to determine if it is a new device or it has been removed. Device IDs are based upon the device and port that it is plugged into. Changing the port will most likely cause the id to change. Plugging the device back into the same port will most likely result in the same device id although this is not guaranteed.

kr_Initialise

Initialises the driver library and sets the error call back function.

Prototype

int kr_Initialise(ErrorCallback errorCallbackFunc, void *pUserData);

Parameters

Name	Type	Input / Output
errorCallbackFunc	ErrorCallback (function ptr)	Input
pUserData	void *	Input

Description

Name	Description
errorCallbackFunc	A function to call whenever an error occurs
pUserData	A pointer to application defined data that will automatically be passed into the callback function

Return Value

1 on success or 0 on error

Remarks

Once initialised devices can be enumerated immediately. If a device callback has been set using SetDeviceChangedCallback and devices are connected to the PC prior to calling this method then it will be raised before this method returns.

kr_Destruct

Clean up the driver library on application exit.

Prototype

int kr_Destruct();

Parameters

None

Description

None

Return Value

None

Remarks

Call on application exit or once you have finished using the library to ensure that resources used by the library have been released.

kr_GetNextDetector

Enumerate the attached detectors. Pass 0 to get the first detector ID and then the value of the previous detector to move through the list. Returns 0 at the end of the list.

Prototype

unsigned int kr_GetNextDetector(unsigned int detectorID);

Parameters

Name	Type	Input / Output
detectorID	Unsigned int	input

Description

Name	Description
detectorID	The id of the previous detector. Pass 0 in to get the first detector

Return Value

Returns the ID of the next detector in the list or 0 if no more detectors are connected.

kr_SetDataReceivedCallback

Register a function that will be called when data is received from a device. Make sure you call BeginDataAcquisition to start reading the data from a device.

Prototype

void kr_SetDataReceivedCallback(DataReceivedCallback callbackFunc, void *pUserData);

Parameters

Name	Type	Input / Output
callbackFunc	DataReceivedCallback (function ptr)	Input
pUserData	void*	Input

Description

Name	Description
callbackFunc	A function to call when any new data is received
pUserData	A pointer to application defined data that will automatically be passed into the callback function.

Return Value

None

Remarks

The data call back function is called whenever data is received from any of the active devices. The call back occurs in a number of separate threads from your main application so you will need to ensure your call back function is thread safe.

Kr_BeginDataAcquisition

Begin acquiring data for a fixed length of time.

Prototype

int kr_BeginDataAcquisition(unsigned int deviceID, unsigned int realTime, unsigned int liveTime);

Parameters

Name	Type	Input / Output
deviceID	Unsigned int	Input
realTime	Unsigned int	Input
liveTime	Unsigned int	Input

Description

Name	Description
deviceID	Id of the device (as returned in kr_GetNextDetector) to begin acquiring data from
realTime	Realtime length in milliseconds to acquire data for or 0 to disable
liveTime	Live time length in milliseconds to acquire data for or 0 to disable

Return Value

ERROR_OK on success or appropriate error code on failure

Remarks

This command instructs a device to start receiving data. If a previous acquisition was performed then this call will append the counts to the previous acquisitions data.
Some errors from this call may be passed via the error call back function specified in kr_Initialise.

The actual data from the device will be returned to the calling application in two ways. If the call back function has been set by kr_SetDataReceivedCallback then the specified call back function will be called as soon as data is received from the device.
Alternatively the data can be read by calling kr_GetAcquiredData which will return all data stored for the device. See kr_GetAcquiredData, kr_GetAcquiredDataEx and kr_ClearAcquiredData for more information.
The live and real time parameters allow the calling application to set a limit to the amount of time to collect data. If both a live time and real time are supplied the detector will acquire data up until the first of the two timers expire. Either (or both) values can be entered as 0 to be disabled. Supplying 0 for both the real time and live time will cause the detector to continue acquiring data indefinitely until stopped with kr_StopDataAcquisition.
The OnError callback function will be called with the `ERROR_ACQUISITION_COMPLETE` error code once either of the live or real time have expired and the acquisition has stopped.

kr_StopDataAcquisition

Stop data acquisition on the specified device.

Prototype

int kr_StopDataAcquisition(unsigned int deviceID);

Parameters

Name	Type	Input / Output
deviceID	Unsigned int	Input

Description

Name	Description
deviceID	Id of the device (as returned in kr_GetNextDetector) to stop acquiring data from

Return Value

ERROR_OK on success or appropriate error code on failure

Remarks

Sends a request to stop the data acquisition for a device. Due to the threaded nature of the driver the device will not be stopped immediately. This means that in some cases new data for the specified device may still be received for a short time following this call. The OnError callback function will be called with the `ERROR_ACQUISITION_COMPLETE` once all data has been received.

Kr_GetAcquiredData

Retrieve the latest set of acquired data for the device.

Prototype

int kr_GetAcquiredData (unsigned int deviceID, unsigned int *pBuffer ,unsigned int *pTotalCounts, unsigned int *pRealTime, unsigned int *pLiveTime);

Parameters

Name	Type	Input / Output
deviceID	Unsigned int	Input
pBuffer	Unsigned int*	Input/Output

pTotalCounts	Unsigned int *	Output
pRealTime	Unsigned int *	Output
pLiveTime	Unsigned int *	Output

Description

Name	Description
deviceID	Id of the device (as returned in kr_GetNextDetector)
pBuffer	An array buffer to copy the data into. The array must be of the size TOTAL_RESULT_CHANNELS. This value can also be NULL to not retrieve any data.
pTotalCounts	An unsigned integer value to return the total event counts recorded by the detector. Can be set to NULL.
pRealTime	An unsigned integer value to return the real time for the scan data. Can be set to NULL.
pLiveTime	An unsigned integer value to return the live time for the scan data. Can be set to NULL

Return Value

ERROR_OK on success or appropriate error code on failure

Remarks

Retrieve the data and total event counts stored for the detector.
If pBuffer is not NULL then channel data is returned into the passed pointer. pBuffer must be an array of at least TOTAL_RESULT_CHANNELS elements. The returned data represents the number of counts recorded on each channel of the detector.
The total counts recorded across all channels on the detector is returned in pTotalCounts.

Kr_GetAcquiredDataEx

Retrieve the latest set of stored data for the device.

Prototype

int kr_GetAcquiredDataEx (unsigned int deviceID, unsigned int *pBuffer ,unsigned int *pTotalCounts, unsigned int *pRealTime, unsigned int *pLiveTime, unsigned int flags);

Parameters

Name	Type	Input / Output
deviceID	Unsigned int	Input
pBuffer	Unsigned int*	Input/Output
pTotalCounts	Unsigned int *	Output
pRealTime	Unsigned int *	Output
pLiveTime	Unsigned int *	Output
Flags	Unsigned int	Input

Description

Name	Description
deviceID	Id of the device (as returned in kr_GetNextDetector)
pBuffer	An array buffer to copy the data into. The array must be of the size TOTAL_RESULT_CHANNELS. This value can also be NULL to not retrieve any data.

pTotalCounts	An unsigned integer value to return the total event counts recorded by the detector. Can be set to NULL.
pRealTime	An unsigned integer value to return the real time for the scan data. Can be set to NULL.
pLiveTime	An unsigned integer value to return the live time for the scan data. Can be set to NULL
flags	A set of flags to enable advanced options

Return Value

ERROR_OK on success or appropriate error code on failure

Remarks

Retrieve the data and total event counts stored for the detector.
When pBuffer is not null channel data is returned into an unsigned int[TOTAL_RESULT_CHANNELS] that is passed into the function. The returned data represents the number of counts recorded on each channel of the detector.
The total counts recorded across all channels on the detector is returned in pTotalCounts.
The additional flag parameter can be used to give the additional options:

- GAD_CLEAR_COUNTS – Clear all data for the device after returning.

Passing a flag parameter of 0 is the same as calling kr_GetAcquiredData()

Kr_ClearAcquiredData

Clear the stored data for a detector.

Prototype

int kr_ClearAcquiredData (unsigned int deviceID);

Parameters

Name	Type	Input / Output
deviceID	Unsigned int	Input

Description

Name	Description
deviceID	Id of the device (as returned in kr_GetNextDetector)

Return Value

ERROR_OK on success or appropriate error code on failure

Remarks

Clear any stored data for the given detector. Real time and live time will also be cleared.

kr_IsAcquiringData

Determine if the given detector is currently acquiring data

Prototypes

int kr_IsAcquiringData(unsigned int deviceID);

Parameters

Name	Type	Input / Output

deviceID	Unsigned int	Input

Description

Name	Description
deviceID	Id of the device (as returned in kr_GetNextDetector)

Return Value

1 if the device is currently acquiring data. 0 if it is not.

kr_GetDeviceName
kr_GetDeviceManufacturer
kr_GetDeviceSerial

Retrieve properties for a device.

Prototypes

int kr_GetDeviceName (unsigned int deviceID, char *pBuffer, int bufferSize, int *pNumBytesOut);
int kr_GetDeviceManufacturer (unsigned int deviceID, char *pBuffer, int bufferSize, int *pNumBytesOut);
int kr_GetDeviceSerial (unsigned int deviceID, char *pBuffer, int bufferSize, int *pNumBytesOut);

Parameters

Name	Type	Input / Output
deviceID	Unsigned int	Input
pBuffer	Char*	Input/Output
bufferSize	Int	Input
pNumBytesOut	Int*	Output

Description

Name	Description
deviceID	Id of the device (as returned in kr_GetNextDetector)
pBuffer	A buffer to copy the returned property string into.
bufferSize	Size of pBuffer in bytes
pNumBytesOut	Number of bytes returned in pBuffer

Return Value

ERROR_OK on success or appropriate error code on failure

kr_GetDeviceVendorID
kr_GetDeviceProductID

Retrieve properties for a device.

Prototypes

int kr_GetDeviceVendorID (unsigned int deviceID, int *pVal);
int kr_GetDeviceProductID (unsigned int deviceID, int *pVal);

Parameters

Name	Type	Input / Output
deviceID	Unsigned int	Input
pVal	Int*	Output

Description

Name	Description
deviceID	Id of the device (as returned in kr_GetNextDetector)
pVal	Variable to return the value into

Return Value

ERROR_OK on success or appropriate error code on failure

kr_SendInt8ConfigurationCommand

Set an 8bit configuration value on the detector

Prototypes

Int kr_SendInt8ConfigurationCommand(unsigned int deviceID, ConfigurationCommandsEnum commandNumber, unsigned char command);

Parameters

Name	Type	Input / Output
deviceID	Unsigned int	Input
commandNumber	ConfigurationCommandsEnum	Input
command	Unsigned char	Input

Description

Name	Description
deviceID	Id of the device (as returned in kr_GetNextDetector)
commandNumber	Enum value identifying which configuration parameter you wish to set
command	Value to set for the parameter

Return Value

ERROR_OK on success or appropriate error code on failure

Remarks

Kr_SendInt8ConfigurationCommand can set a number of values on the detector. The command parameter is the actual value to set. Please be careful as changing these settings can cause serious problems to the configuration of your detector including settings that were set up by the Kromek team before the detector was shipped.

Valid Commands are:

```
HIDREPORTNUMBER_SETGAIN (all detectors)
HIDREPORTNUMBER_SETPOLARITY (K102 only)
```

Valid values for k102 polarity:

```
HIDREPORTDATA_SETPOLARITY_POSITIVE
HIDREPORTDATA_SETPOLARITY_NEGATIVE
```

kr_SendInt16ConfigurationCommand

Set an 16 bit configuration value on the detector

Prototypes

Int kr_SendInt16ConfigurationCommand(unsigned int deviceID, ConfigurationCommandsEnum commandNumber, unsigned short command);

Parameters

Name	Type	Input / Output
deviceID	Unsigned int	Input
commandNumber	ConfigurationCommandsEnum	Input
command	Unsigned short	Input

Description

Name	Description
deviceID	Id of the device (as returned in kr_GetNextDetector)
commandNumber	Enum value identifying which configuration parameter you wish to set
command	Value to set for the parameter

Return Value

ERROR_OK on success or appropriate error code on failure

Remarks

Kr_SendInt16ConfigurationCommand can set a number of values on the detector. The command parameter is the actual value to set. Please be careful as changing these settings can cause serious problems to the configuration of your detector including settings that were set up by the Kromek team before the detector was shipped.

Valid Commands are:

```
HIDREPORTNUMBER_SETLLD
HIDREPORTNUMBER_SETBIAS16 (SIGMA / TN15 only)
```

Appendix A – Example Applications

Two example applications have been provided:

- SpectrometerExample is a full GUI application which supports acquisition from multiple devices simultaneously.
- ConsoleExample is a very basic example of usage for a command line application. This example shows an easy way to test the libraries features but is rather limited in its error reporting and functionality.

To support both Windows and Linux the SpectrometerExample makes use the Qt developer libraries and tools (http://www.qt.io)

Please read *Important Development Note* in the section '*Using the Application Level Driver'* for important information on how to build and run the applications.