Description

This package provides a set of functions to control the VariSpec filter, which may be called from C or C++ programs. It incorporates all aspects of the filter communication, including low-level serial routines. With these routines, one can address the filter as a virtual object, with little need for detailed understanding of its behavior. This simplifies the programming task for those who want to integrate the VariSpec into larger software packages.

File manifest
All files are contained in a single installer file which includes the following:

vsdrvr.h	declaration file
vsdrvr.lib	library stub file
vsdrvr.dll	run-time library
vsdrvr_r1p37.pdf	(this file) release notes and programmer's guide

{sample program using VsDrvr package}
registryAccess.cpp
registryAccess.h
resource.h
stdafx.h
VsConfigDlg.cpp
VsConfigfDlg.h
VsGui.cpp
VsGui.h
VsGui.mak
VsGui.rc
VsGuiDlg.cpp
VsGuiDlg.h
VsSweep.cpp
VsSweep.h

Development cycle
In order to use the DLL, one should take the following steps:

a) Add #include "vsdrvr.h" statements to all files that access the VariSpec software
b) Add vsdrvr.lib to the list of modules searched by the linker
c) Place a copy of vsdrvr.dll in either the folder that includes the executable code for the program being developed; or, preferably, in the windows system folder.

Failures in step a) will lead to compiler errors; in step b) to linker errors; in step c) to a run-time error message that "a required .DLL file, vsdrvr.dll, was not found".

VariSpec filter configuration

The VariSpec filter communicates via ASCII commands sent over an RS-232 interface or USB. The RS232 can operate at 9600 or 19,200 baud, while the USB appears as a virtual COMx device. While it appears to operate at either 9600 baud or 115.2 kbaud , the actual data transmission occurs at 12 MBaud over the USB.

Each command is terminated with an end-of-line terminator which can be either a carriage-return <c/r> or line feed <l/f>.

For RS-232 models, the baud rate and terminator character are selected using DIP switches inside the VariSpec electronics module. Default settings are 9600 baud, and the <c/r> character (denoted '\r' in the C language).

For USB devices, the terminator is always <c/r>.

For latest information, or to determine how to alter the settings from the factory defaults, consult the VariSpec manual.

Timing and latency

The VariSpec filter takes a finite time to process commands, which adds an additional delay to that imposed by simple communication delays. In general, the time to process a given command is short except for the following operations:

- filter initialization
- wavelength selection
- palette definition

The first of these is quite lengthy (30 seconds or more) because it involves measurements and exercising of the liquid crystal optics. The latter two are much faster but still can take a significant amount of time (up to 300 ms) on the older RS-232 electronics due to the computations involved. On the newer, USB electronics, the latter two functions are completed in less than 2 ms, so there is little speed benefit to using palette tables on these systems.

Because of the processing time, the functions that handle these actions offer the option of waiting until the action is complete before returning (so-called synchronous operation); although they can be called in an asynchronous mode where the function returns as soon as all commands have been sent to the VariSpec, without waiting for them to run to completion.

Another option is to use implicit palette tables. If this is enabled, by calling the VsEnableImplicitPalette() function, the driver will define the settings for a given wavelength once, then saves the results within the VariSpec for faster access next time that wavelength is used. Subsequent access times are essentially instantaneous, until either all of the 128 palette states are in use, or the palette is cleared via the VsClearPalette() command.

The VsIsReady() function can be used to determine whether a filter is done processing all commands. Ideally, one should check VsIsReady() using a timer or the like to wait efficiently, so that the host PC is free to do other tasks while waiting for the VariSpec.

The VariSpec always processes each command to completion before starting on the next command, and it has a 256 byte input buffer, so there is no problem issuing several commands at once; they will all be executed, and in the order given.

This also indicates another way to coordinate one's program with the VariSpec filter: one can issue any of the VsGet*xxx*() functions, which query the filter. Since these do not return until the filter has responded, one may be sure that there are no pending commands when the VsGet*xxx*() function completes.

The VsDrvr package provides for automatic re-try of commands up to 3 times, in the event that communications are garbled, and will wait up to 2 seconds for completion of serial commands. The number of retries can be set from 0 to 10, and the latency adjusted, if desired. However, there should be no need to do so. The hardware and software have been tested and observed to execute several million commands without a single communications error, so in practice the need for the retry protocol is very slight. Communication speed is not improved by reducing the latency, since commands proceed when all characters are received, and the latency time to time-out is only relevent when there is a communications lapse – and as noted, these are very unlikely so the performance burden of retries should not be a practical consideration.

Multiple Filters and Multiple Processes

The VsDrvr now supports multiple VariSpec filters per process. However, it is not possible for several active processes seek to control the same filter at the same time.

Each filter instance is associated with an integer handle assigned during a successful call to VsOpen, which is passed as an argument to all other filter control functions. This handle is checked, and the correct handle must be used in all calls.

Multiple threads can be used within a process, provided that only one thread talks with the VariSpec at any instant. Typically, one calls VsOpen in the main thread, and then passes the filter handle to worker (or acquisition) threads. Again, it is the software designer's responsibility to make sure that only one thread at a time accesses the filter.

Program flow and sequence

Typical programs should use the following API calls

(all applications, upon initiating link to the filter)

- call VsOpen() to establish communications link (required)
- call VsIsPresent() to confirm a filter is actually present
- call VsIsReady() in case filter is still doing power-up sequence

<wait until no longer busy>

- call VsGetFilterIdentity() to learn wavelength limits and serial number if needed

(if setting wavelengths via implicit palettes; recommended especially with older filters)

- call VsEnableImplicitPalettes()
-

(to set wavelengths, either directly or via implicit palettes)

- call VsSetWavelength() and VsGetWavelength() to select and retrieve tuning

(if setting wavelengths by means of palettes, and managing palettes explicity)

- call VsDefinePaletteEntry() and VsClearPalette() to define palette entries
- call VsSetPalette() and VsGetPalette() to select and retrieve palette state

(all applications, when done with the filter)

- call VsClose() to release the communications link (required)

Sample program

Source code for a sample program, VsGui, is provided, which illustrates how to control a VariSpec filter using the VsDrvr package. All filter control code lives in the VsGuiDlg.cpp module, specifically in the Connect(), RequestToSetWavelength(), and VsWriteTimerProc() functions. The latter two use a system timer to decouple the GUI from the actual filter control, for more responsive feedback to the user. Such an approach is unnecessary if palettes are used, which is preferable when one wishes the best real-time performance. See the VariSpec manual for further information.

Auxiliary commands

Certain commands are normally only used at the factory when filters are being built and configured, or in specialized configurations. These appear after the normal command set in the listing below.

Obsolescent commands

The VsConnect(), VsIsConnected(), and VsDisconnect() functions are obsolescent. They are supported in this release, but will not necessarily exist in releases after 1.3x.

As they are obsolescent, they are not recommended for new code. These function calls are not documented further in this manual.

Summary of commands

Normal Commands
VsClearError(vsHnd)
VsClearPalette(vsHnd)
VsClearPendingCommands(vsHnd)
VsClose(vsHnd)
VsDefinePalette(vsHnd, palEntry, wl)
VsEnableImplicitPalette(vsHnd, isEnabled)
VsGetError(vsHnd, *pErr)
VsGetFilterIdentity(vsHnd, *pVer, *pSerno, *pminWl, *pmaxWl)
VsGetMode(vsHnd, int *pMode)
VsGetPalette(vsHnd, *ppalEntryNo)
VsGetSettleMs(vsHnd, *psettleMs)
VsGetTemperature(vsHnd, *pTemperature)
VsGetWavelength(vsHnd, *pwl)
VsGetWavelengthAndWaves(vsHnd, double *pWl, double *pwaves)
VsGetWavelengthStep(vsHnd, double *pWlStep)
VsGetWaveplateLimits(vsHnd, double *pminWaves, double *pmaxWaves)
VsGetWaveplateStages(vsHnd, int *pnStages)
VsIsPresent(vsHnd)
VsIsReady(vsHnd)
VsOpen(*pvsHnd, portName, *pErrorCode)
VsSetLatencyMs(vsHnd, nLatencyMs)
VsSetMode(vsHnd, mode)
VsSetPalette(vsHnd, palEntry)
VsSetRetries(vsHnd, nRetries)
VsSetWavelength(vsHnd, wl, confirm)
VsSetWavelengthAndWaves(vsHnd, wl, waveplateVector)
VsSetWavelengthStep(vsHnd, double wl, BOOL confirm)

Auxiliary commands
VsGetAllDrive(vsHnd, *pStages, drive[])
VsGetNstages(vsHnd, *pStages)
VsGetPendingReply(vsHnd, reply, nChars, *pQuit, firstMs, subsequentMs)
VsGetReply(vsHnd, reply, nChars, waitMs)
VsIsDiagnostic(vsHnd)
VsIsEngagedInBeam(vsHnd)
VsSendBinary(vsHnd, bin[], nChars, clearEcho)
VsSendCommand(vsHnd, cmd, sendEolChar)
VsSetStageDrive(vsHnd, stage, drive)
VsThermistorCounts(vsHnd, *pCounts)
VsSetWavelengthWavesConfirm(vsHnd, BOOL confirm)

Alphabetical list of function calls

Syntax
Throughout this manual, the following conventions are used:

```
VSDRVR_API      Int32 VsOpen(
                        VS_HANDLE *vsHnd,
                        LPCSTR port,
                        Int32 *pErrorCode
                        )
```

Bold text is used for function names
Italics indicate variables whose names (or values) are supplied by the user in their code

Name-mangling
The declaration file vsdrvr.h includes statements that render the API names accurately in a C++ environment, i.e. free of the name-mangling decoration suffix that is normally added by C++ compilers. Thus the functions can be called freely from either C or C++ programs, using the names exactly as shown in this manual or in the VsDrvr.h file.

Call and argument declarations
The call protocol type, VSDRVR_API, is declared in vsdrvr.h, as are the types Int32 and VS_HANDLE.

Errors
All functions return an Int32 status value, which is TRUE if the routine completed successfully and FALSE if there was an error.

If there is an error in the VsOpen() function, the error is returned in *pErrorCode.

If there is an error in communicating with a filter after a successful VsOpen(), one should use the VsGetError() function to obtain the specific error code involved. This function returns VSD_ERR_NOERROR if there is no error pending.

Main and auxiliary functions
The next section provides a description of the main functions, in alphabetic order; followed by the auxiliary functions, also in alphabetical order. In normal use, one will probably have no need for the auxiliary functions, but this list is provided for completeness.

VSDRVR_API Int32 **VsClearError**(
VS_HANDLE *vsHnd*
)

Arguments:
vsHnd handle value returned by VsOpen()

Purpose: this function clears any pending error on the VariSpec. This resets the error LED on the filter, and sets the pending error to VS_ERR_NOERROR.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsClearPalette**(
VS_HANDLE *vsHnd*
)

Arguments:
vsHnd handle value returned by VsOpen()

Function: clears all elements of the current filter palette and renders the current palette element undefined.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsClearPendingCommands**(
VS_HANDLE *vsHnd*
)

Arguments:
vsHnd handle value returned by VsOpen()

Function: clears all pending commands including any presently in-process

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsClose**(
VS_HANDLE *vsHnd*
)

Arguments:
vsHnd handle value returned by VsOpen(). May also be NULL, in which case all VariSpec filters are disconnected.

Function: Disconnects the filter.

Returns: TRUE if successful, FALSE otherwise

Notes: No other functions will work until VsOpen() is called to re-establish communications with the filter.

VSDRVR_API Int32 **VsDefinePalette**(
VS_HANDLE *vsHnd*,
Int32 *palEntry*,
double *wl*)

Arguments:

vsHnd	handle value returned by VsOpen()
palEntry	palette entry to be defined, in the range [0, 127]
wl	wavelength associated with this palette entry

Function: creates a palette entry for the entry and wavelength specified. This palette entry can then be accessed using VsSetPalette() and VsGetPalette() functions.

Returns: TRUE if successful, FALSE otherwise

Notes: palettes provide a fast way to define filter settings for wavelengths that are to be repeatedly accessed. The calculations are performed once, at the time the palette element is defined, and the results are saved in a palette table to tune to that wavelength without repeating the underlying calculations. And, one may cycle through the palette table, once defined, by means of TTL a trigger signal to the filter electronics.

For more information about using palettes, consult the VariSpec user's manual.

VSDRVR_API Int32 **VsEnableImplicitPalette**(
VS_HANDLE *vsHnd*,
BOOL *imlEnabled*)

Arguments:

vsHnd	handle value returned by VsOpen()
implEnabled	selects whether to use implicit palette definition

Function: enables or disables implicit palette generation when wavelengths are defined using the VsSetWavelength function. If enabled, a new palette entry is created whenever a new wavelength is accessed, and the VsSetWavelength function will use this palette entry whenever that wavelength is accessed again, until the palette is cleared. The result is improved tuning speed; however, it means that the palette contents are altered dynamically, which can be a problem if one relies upon the palette contents remaining fixed.

Clearing the palette with VsClearPalette() will clear all implicit palette entries as well as explicitly defined palette entries. This is useful if one knows that wavelengths used previously will not be used again, or that a new set of wavelengths is about to be defined and one wishes to make sure there is sufficient room in the palette.

Returns: TRUE if successful, FALSE otherwise

Notes: By default, the implicit palette is enabled for VariSpec filters that have RS-232 interface, and is disabled for newer VariSpec filters that have the USB interface. This is because the newer filters perform the filter tuning calculations fast enough that no performance improvement is obtained by using the implicit palette to set wavelength.

For more information about using palettes, consult the VariSpec user's manual.

VSDRVR_API Int32 **VsGetError**(
VS_HANDLE *vsHnd*,
Int32 **pErr*)

Arguments:

vsHnd	handle value returned by VsOpen()
pErr	pointer to the int that will receive the most recent error code

Purpose: this function clears any pending error on the VariSpec. This resets the error LED on the filter, and sets the pending error to VS_ERR_NOERROR.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsGetFilterIdentity**(
VS_HANDLE *vsHnd*,
Int32 **pVer*,
Int32 **pSerno*,
double **pminWl*,
double **pmaxWl*
)

Arguments:

vsHnd	handle value returned by VsOpen()
pVer	pointer to variable that receives the filter firmware version
pSerno	pointer to variable that receives the filter serial number
pminWl	pointer to variable that receives the filter's minimum wavelength
pmaxWl	pointer to variable that receives the filter's maximum wavelength

Purpose: this function reads the filter's information using the VariSpec 'V' command, and puts it to the call variables. Any one of the pointers may be NULL, in which case that piece of information is not returned.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsGetMode**(
VS_HANDLE *vsHnd*,
Int32 **pMode*
)

Arguments:

vsHnd	handle value returned by VsOpen()
pMode	pointer to variable that receives the filter mode

Purpose: this function enables one to read the filter's present mode. The mode describes how the filter responds to hardware triggers, and is described in the filter manual.

If the pointer *pMode is NULL, no information is returned.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsGetPalette**(
VS_HANDLE *vsHnd*,
Int32 **ppalEntry*
)

Arguments:

vsHnd	handle value returned by VsOpen()
ppalEntry	pointer to int that receives the 0-based palette entry number. This pointer may not be NULL.

Purpose: this function determines what palette entry is currently active and returns it to **ppalEntry*. If the present palette entry is undefined, it sets **ppalEntry* to –1 and returns a successful status code.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsGetSettleMs**(
VS_HANDLE *vsHnd*,
Int32 **pSettleMs*
)

Arguments:

vsHnd	handle value returned by VsOpen()
pSettleMs	pointer to variable that receives the filter settling time

Purpose: this function returns the filter's settling time, in milliseconds. This is useful for establishing overall system timing. The settling time is defined as beginning at the moment that the electronics have processed the request to change wavelength, as determined by VsIsReady() or equivalent. At that moment, the new set of drive signals are applied to the optics, and the optics will settle in **psettleMs* milliseconds.

The settling time is defined as a 95% settling time, meaning the filter has settled to 95% of its ultimate transmission value at the new wavelength being tuned to.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsGetTemperature**(
VS_HANDLE *vsHnd*,
double **pTemperature*
)

Arguments:

vsHnd	handle value returned by VsOpen()
pTemperature	pointer to double that will receive the filter temperature, in C This pointer may not be NULL

Purpose: this function determines the filter temperature using the VariSpec 'Y' command, and puts the result to *pTemperature.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsGetWavelength**(
VS_HANDLE *vsHnd*,
double **pwl*
)

Arguments:

vsHnd	handle value returned by VsOpen()
pwl	pointer to double that will receive the filter wavelength, in nm This pointer may not be NULL

Purpose: this function determines the current filter wavelength and returns it to **pwl*. If the present wavelength is undefined, it sets **pwl* to –1 and returns a successful status code.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsGetWavelengthAndWaves**(
VS_HANDLE *vsHnd*,
double **pwl*,
double **pwaves*
)

Arguments:

vsHnd	handle value returned by VsOpen()
pwl	pointer to double that will receive the filter wavelength, in nm. This pointer may not be NULL
pwaves	pointer to double array that will receive one or more waveplate settings. The actual number of settings may be determined by VsGetWaveplateStages().

Purpose: this function determines the current filter wavelength and returns it to **pwl*. If the present wavelength is undefined, it sets **pwl* to –1 and returns a successful status code. If the present wavelength is defined, it also returns the waves of retardance at each of the polarization analysis waveplates in the optics, in the pwaves[] array.

Returns: TRUE if successful, FALSE otherwise

Notes: See the description of the VsGetWaveplateStages() command for more detail on what stages are considered waveplates.

VSDRVR_API Int32 **VsGetWaveplateLimits(**
VS_HANDLE *vsHnd,*
double **pminWaves,*
double **pmaxWaves*
)

Arguments:
vsHnd handle value returned by VsOpen()
pminWaves pointer to double array that will receive the minimum retardances possible at each of the waveplate stages in the filter optics.
pmaxWaves pointer to double array that will receive the maximum retardances possible at each of the waveplate stages in the filter optics

Purpose: this function determines the range of retardances that are possible at each waveplate stage, in waves, at the present wavelength setting. Note that the retardance range is itself a function of wavelength, so the results will vary as the wavelength is changed.

Returns: TRUE if successful, FALSE otherwise

Notes: See the description of the VsGetWaveplateStages command for more detail on what stages are considered waveplates.

VSDRVR_API Int32 **VsGetWaveplateStages(**
VS_HANDLE *vsHnd*,
Int32 **pnwpStages*
)

Arguments:
vsHnd handle value returned by VsOpen()
pnwpStages pointer to Int32 that will receive the number of waveplate stages in the filter optics. This pointer may not be NULL

Purpose: this function determines how many polarization analysis stages are present in the optics and returns this number. Note that although all VariSpec filters operate by means of variable retarder element, optical stages that perform wavelength tuning rather than polarization analysis are not treated as waveplate stages.

For example, most VariSpec filters do not include any polarization analysis stages and thus report no waveplates. VsGetWaveplateStages will return a value of 2 for conventional PolScope optics.

In contrast, VsGetNstages() reports the total number of stages in a filter, including stages that perform polarization analysis and stages that perform wavelength tuning.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsGetWavelengthStep**(
VS_HANDLE *vsHnd*,
double **pwlStep*
)

Arguments:

vsHnd	handle value returned by VsOpen()
pwlStep	pointer to double that will receive the wavelength step, in nm This pointer may not be NULL

Purpose: this function determines the wavelength step that will be applied for wavelength-increment operations, and returns it to **pwlStep*. If the present wavelength is undefined, it returns a successful status code.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsIsPresent**(
VS_HANDLE *vsHnd*
)

Arguments:
vsHnd handle value returned by VsOpen()

Function: determines whether a filter is actually present and responding. This is done using the status-check character '!' as described in the VariSpec manual.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsIsReady**(
VS_HANDLE *vsHnd*
)

Arguments:
vsHnd handle value returned by VsOpen()

Function: determines whether the filter is done processing all commands, and is ready to receive new commands.

Returns: TRUE if successful, FALSE otherwise

Notes: this is useful when sending commands such as VsSetWavelength(), VsInitialize(), VsExercise(), and VsDefinePaletteEntry() in asynchronous mode. These commands take a prolonged time, and running them synchronously ties up the processor waiting. Alternatively, one can create a loop that uses CreateWaitableTimer(), SetWaitableTimer(), and WaitForSingleObject() to call VsIsReady() at intervals, checking whether the filter is ready. This approach, though more work for the programmer, leaves most of the processor capacity free for other tasks such as GUI update and the like.

VSDRVR_API Int32 **VsOpen** (VS_HANDLE *p*vsHnd*,
LPCSTR *port*,
Int32 **pErrorCode*
)

Arguments:

pvsHnd	pointer to handle. This pointer may not be NULL.
port	port name, such as "COM1"
pErrorCode	pointer to Int32 to receive an error code if VsOpen() fails

Purpose: establishes a connection to the VariSpec using the port specified, and automatically determines the baud rate and end-of-line character for subsequent communications. It also retrieves the filter's serial number and wavelength range, to confirm that it is a VariSpec and not some other similar device. However, these are retrieved purely as an integrity check, and the values are not returned to the calling application. See VsGetFilterInfo() to access this information.

If the device responds as a VariSpec does when it is not ready (i.e. still initializing), VsOpen() fails and returns the error code VSD_ERR_BUSY. However, one may not be sure that the device is a VariSpec until VsOpen() completes successfully

The error codes returned by this function are listed in VsDrvr.h. When VsOpen() runs successfully, *pErrorCode is set to VSD_ERR_NOERROR.

The handle associated with this filter is set by VsOpen() to a nonzero handle value if successful, or to NULL if no connection is established.

The *port* may refer to COM1 through COM8.

Return: TRUE if successful, FALSE otherwise

Notes: Until this function is called, none of the other functions will work.

VSDRVR_API Int32 **VsSetLatency**(
VS_HANDLE *vsHnd*,
Int32 *latencyMs*
)

Arguments:

vsHnd	handle value returned by VsOpen()
latencyMs	the serial port latency, in ms, in the range [1, 5000]

Purpose: this function sets the latency time for USB or RS-232 commands to the value given by *latencyMs*. Commands that do not conclude in this time are considered to have timed-out.

Returns: TRUE if successful, FALSE otherwise

Notes: increasing the latency time does not increase the time for commands to complete, nor does it insert any delays in normal processing. It merely defines the window for maximum transmission time, beyond which time an error is reported.

VSDRVR_API Int32 **VsSetPalette**(
VS_HANDLE *vsHnd*,
Int32 *palEntry*
)

Arguments:

vsHnd	handle value returned by VsOpen()
palEntry	the palette entry to be set, in the range [0, 127]

Purpose: this function sets the filter to the palette entry specified by *palEntry*

Returns: TRUE if successful, FALSE otherwise

Notes: palettes are a good way to control the filter in applications where it will be cycled repeatedly to various, fixed wavelength settings. Palettes calculate the filter settings once, and save the results for rapid access later, rather than calculating them each time, as occurs when one sets the wavelength directly with VsSetWavelength(). See the VariSpec manual for more information on palettes.

VSDRVR_API Int32 **VsSetRetries**(
VS_HANDLE *vsHnd*,
Int32 *nRetries*
)

Arguments:

vsHnd	handle value returned by VsOpen()
nRetries	the number serial communications retries, in the range [0, 10]

Purpose: The VsDrvr software automatically detects errors in communication and re-sends if an error is detected. This function sets the number of times to retry sending any single command, before reporting a communications failure. The default is 3, which should be adequate, and one should rarely need to change this, if ever. The primary purpose of this function is to enable setting the number of retries to zero, to force single-error events to cause detectable errors (as they would normally be fixed automatically via the retry mechanism)

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsSetWavelength**(
VS_HANDLE *vsHnd*,
double *wl*,
BOOL *confirm*
)

Arguments:

vsHnd	handle value returned by VsOpen()
wl	wavelength to tune to, in nm
confirm	logical flag, indicating whether to confirm actual wavelength value

Purpose: this function sets the filter wavelength to the value in *wl*. If *confirm* is TRUE, it waits for the filter to complete the command, and then reads back the actual wavelength to confirm it was implemented successfully. Note that the only time there can be a disparity is when the wavelength requested by *wl* lies outside the legal range for that filter, or if the wavelength is specified to a finer resolution than the filter recognizes (normally, 0.01 nm).

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsSetWavelengthAndWaves**(
VS_HANDLE *vsHnd*,
double *wl*,
double waves[]
)

Arguments:

vsHnd	handle value returned by VsOpen()
wl	wavelength to tune to, in nm
waves[]	array of retardance values, in waves
confirm	logical flag, indicating whether to confirm actual wavelength value

Purpose: for filters which have one or more retardance elements, this function sets the filter wavelength to the value in *wl* and sets the retarders to the values given in *waves[]*. These are done together, so the values for *wl* and *waves[]* must be within the permitted range for the optics, it does not cause an error if either value alone would be in conflict with the filter settings in effect just prior to issuing this command.

By default, this command waits for the filter to complete the command, and then reads back the actual values attained, it was implemented successfully. Note that the only time there can be a disparity is when the wavelength requested by *wl* lies outside the legal range for that filter, or if the wavelength is specified to a finer resolution than the filter recognizes (normally, 0.01 nm).

One may use the command VsSetWavelengthAndWavesConfirm() to eliminate the confirmation step. However, this command is not recommended for VariSpec models which use the RS-232 interface, as there could be undetected errors. Models using the USB interface will run reliably either with confirmation, or without it.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsSetWavelengthWavesConfirm**(
VS_HANDLE hVs,
BOOL confirm
)

Arguments:

vsHnd	handle value returned by VsOpen()
confirm	logical flag, indicating whether to confirm or not, in subsequent calls to VsSetWavelengthAndWaves

Purpose: Enables (disables) a confirmation read-back step when performing VsSetWavelengthAndWaves(). Effectively, this yields synchronous (asynchronous) operation. By default, this command operates synchronously.

Use of asynchronous mode is NOT RECOMMENDED for VariSpec models which use the RS-232 interface, as there could be errors if subsequent commands are issued before the (asynchronous) VsSetWavelengthAndWaves() command completes its action .

Models using the USB interface will run reliably in either mode.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsSetWavelengthStep**(
VS_HANDLE *vsHnd*,
double *wlStep*,
BOOL *confirm*
)

Arguments:

vsHnd	handle value returned by VsOpen()
wlStep	wavelength step, in nm
confirm	logical flag, indicating whether to confirm actual wavelength value

Purpose: this function sets the filter wavelength step to the value in *wlStep*. If *confirm* is TRUE, it waits for the filter to complete the command, and then reads back the actual wavelength step to confirm it was implemented successfully. Wavelength steps may be negative, in which case wavelength-increment operations tune to successively shorter wavelengths. Note that the only time there can be a disparity is when the wavelength step requested by *wl* lies outside the legal range for that filter, or if the wavelength step is specified to a finer resolution than the filter recognizes (normally, 0.01 nm).

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsGetAllDrive**(
VS_HANDLE *vsHnd*,
Int32 *pStages,
Int32 drive[]
)

Arguments:

vsHnd	handle value returned by VsOpen()
pStages	pointer to int that will receive the number of stages in the filter
drive[]	int array to receive the filter drive levels.

Purpose: this function reports the number of filter stages in **pStages*. If this argument is NULL, it is ignored. The function returns the actual drive level at each stage, in counts, in *drive[]* , which must not be NULL.

Returns: TRUE if successful, FALSE otherwise

Notes: The array *drive[]* must be large enough to receive all the drive levels – if the exact number of stages is not known, call VsGetNstages() first, or allocate enough array elements (12) to accommodate the largest filter design.

VSDRVR_API Int32 **VsGetNstages**(
VS_HANDLE *vsHnd*,
Int32 **pStages*
)

Arguments:

vsHnd	handle value returned by VsOpen()
pStages	pointer to int that will receive the number of stages in the filter

Purpose: this function determines the number of optical stages in the filter and returns it in **pStages*, which may not be NULL.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsGetPendingReply**(
VS_HANDLE *vsHnd*,
LPSTR *reply*,
Int32 *nChars*,
Int32 **pQuit*,
Int32 *firstMs*,
Int32 *subsequentMs*
)

Arguments:

vsHnd	handle value returned by VsOpen()
reply	pointer to buffer that is to receive the reply
nChars	number of characters to receive
pQuit	pointer to flag to control this function – see Notes below
firstMs	maximum time to wait, in ms, for first character of reply
subsequentMs	maximum time to wait, in ms, for each subsequent character

Purpose: this function is used to exploit some of the less-common aspects of the filter, and it is likely that most programs will require its use. It receives a reply from the filter that may not arrive for a long time. The routine waits up to *firstMs* for the first character to arrive. Subsequent characters must arrive within *subsequentMs* of one another. Typically, this routine is called with a high value for *firstMs* and a lower value for *subsequentMs*.

Returns: TRUE if successful, FALSE otherwise

Notes: *pQuit* can be used to cancel this function while it is waiting for the reply, if that is desired, such as to respond to a user cancellation request. To use this feature, *pQuit* must be non-NULL and **pQuit* must be FALSE at the time VsGetPendingReply() is called. VsGetPendingReply() checks this address periodically, and if it discovers that **pQuit* is TRUE, it will cancel and return immediately.

VSDRVR_API Int32 **VsGetReply**(
VS_HANDLE *vsHnd*,
LPSTR *reply*,
Int32 *nChars*,
Int32 *waitMs*
)

Arguments:

vsHnd	handle value returned by VsOpen()
reply	pointer to buffer that will receive the filter reply
nChars	the number of characters sought
waitMs	the maximum time, in ms, to wait for the reply

Purpose: this function is used to exploit those filter commands that are not directly provided by other functions, and most programmers will not need to use it. If the reply is not received in the time indicated by *waitMs*, or if less than *nChars* are received, the function returns with an unsuccessful status code.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsIsDiagnostic**(
VS_HANDLE *vsHnd*
)

Arguments:
vsHnd handle value returned by VsOpen()

Function: determines whether the filter is in the diagnostic mode that is used at the factory for setup and calibration. This command is reserved for CRI use only.

Returns: TRUE if diagnostic, FALSE otherwise.

VSDRVR_API Int32 **VsIsEngagedInBeam**(
VS_HANDLE *vsHnd*
)

Arguments:
vsHnd handle value returned by VsOpen()

Function: determines whether the filter is engaged in the beam, when configured into certain CRI systems. This function is reserved for CRI use only

Returns: TRUE if engaged in the beam, FALSE otherwise

VSDRVR_API Int32 **VsSendBinary**(
VS_HANDLE *vsHnd*,
char **bin*,
Int32 *nChars*,
BOOL *clearEcho*
)

Arguments:

vsHnd	handle value returned by VsOpen()
bin	pointer a buffer that contains binary data to be sent to the filter
nChars	the number of binary characters to be sent
clearEcho	flag indicating whether to clear echo characters from the queue

Purpose: this routine sends binary blocks of data to the filter. This is only necessary when programming calibration data to the filter, and it is not anticipated that this function will be necessary in any normal use.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsSendCommand**(
VS_HANDLE *vsHnd*,
LPCSTR *cmd*,
BOOL *sendEolChar*)

Arguments:

vsHnd	handle value returned by VsOpen()
cmd	pointer to the command to be sent to the filter
sendEolChar	flag indicating whether to append the end-of-line character or not

Purpose: this function sends the command in *cmd* to the filter, and appends an end-of-line terminator (or not) based on *sendEolChar*. It automatically retrieves and discards the character echo of this command by the VariSpec. It does not automatically retrieve the reply, if any, from the VariSpec.

Returns: TRUE if successful, FALSE otherwise

Notes: The parameter *sendEolChar* should normally be true in all cases, unless one is sending individual character commands such as the '!' or '@' commands described in the VariSpec user's manual.

VSDRVR_API Int32 **VsSetStageDrive**(
VS_HANDLE *vsHnd*,
Int32 *stage*,
Int32 *drive*
)

Arguments:

vsHnd	handle value returned by VsOpen()
stage	stage number whose drive level is to be adjusted
drive	drive level, in counts, for that stage

Purpose: this function provides a way to manually adjust the drive levels at each of the filter's optical stages. It is normally used only during manufacture, and is not a function that most software programs will have any reason to use.

Returns: TRUE if successful, FALSE otherwise

Notes: none

VSDRVR_API Int32 **VsThermistorCounts**(
VS_HANDLE *vsHnd,*
Int32 **pCounts*
)

Arguments:

vsHnd	handle value returned by VsOpen()
pCounts	pointer to int that will receive the thermistor signal, in counts

Purpose: this function provides a way to determine the signal level, in counts, at the thermistor. It is normally used only during manufacture, and is not a function that most software programs will have any reason to use.

Returns: TRUE if successful, FALSE otherwise

Notes: none