

Table of Contents

CDI Spectrograph Card Driver Function Library 32-bit DLL for Windows	2
CDI_Init.....	3
CDI_Convert	5
CDI_TimerEnable	6
CDI_UpdateData	6
CDI_SetShowoffFlag	6
CDI_ChangeOffset.....	7
CDI_ChangeIntTime	7
CDI_SetCCDTemp.....	7
CDI_ChangeCompensation.....	8
CDI_ChangeTriggerMode.....	8
CDI_ChangeBoardAddr.....	9
CDI_GetIntegrateTime.....	9
CDI_GetTriggerMode	10
CDI_GetCompensation	10
CDI_LinearizeLive.....	11
CDI_SetupDualbeam.....	11
CDI_OnStoreCopylive	11
CDI_OnDFStoreCopylive	12
CDI_Rereference	13
CDI_Acqbg	13
CDI_Acqref.....	13
CDI_SaveSetupFile.....	13
CDI_LoadEESetup.....	14
CDI_SaveEESetup	14
CDI_SetupColorimetry.....	15
CDI_GetColorimetry.....	15
CDI_SetupAveraging	15
CDI_ResetAverage.....	16
CDI_GetAvgSamples.....	16
CDI_GetAvgN.....	16
CDI_GetNumPtsLive	16
CDI_SaveDataFile.....	16
CDI_TriggerBurst.....	17
CDI_SetupBurstMode	17
CDI_GetDualbeamMode.....	17
CDI_GetRefBgStatus	18
CDI_ShowMessages.....	18
CDI_OpticSwitch	18
CDI_OpticSwitch2	18
CDI_ReadUnitSerNo.....	19
CDI_ChangeGain	19
CDI_InputPixelData	19
CDI_OutputPixelData	20
CDI_ReadData	20
CDI_ChangeNoBoard	21
CDI_SetupColorimetryEx	21
CDI_GetColorimetryEx.....	22
Appendix A - CDI32.H Header File.....	Error! Bookmark no

CDI Spectrograph Card Driver Function Library 32-bit DLL

The library functions are supplied in a Windows™ 32-bit Dynamic Link Library (DLL). These software drivers for embedded applications include most of the functions provided by the CDI SPEC stand-alone windows application, including:

- * Background subtraction
- * Percent Transmittance / Reflectance
- * Transmittance / Reflectance in dB
- * Absorbance Units (AU)
- * Absolute calibration to an ideal blackbody source (for example, a tungsten-halogen lamp of a known color temperature)
- * User defined radiance calibration for use of NIST traceable lamp calibration data (ASCII file)
- * Color Analysis - Tristimulus X, Y, Z, chromaticity x, y, L*, a*, b*, u*, v*, hue angle, saturation, chroma, and delta E* color difference, color temperature
- * Linearization / Interpolation
- * Sample Averaging / co-add
- * Savitsky-Golay Smoothing
- * Acquire and Store Reference and Background Samples

Individual functions are also available for accessing all of the instrument controls. The entire spectrograph driver can be embedded in an application using as few as two DLL functions -- `CDI_Init()` and `CDI_Convert()`. The `CDI_Init()` function reads the ASCII instrument setup files stored by the CDI SPEC Windows application. This allows the user to set up all instrument parameters using the graphic user interface of the SPEC Windows program, and save the setup for use by the DLL driver functions. Simply call the `CDI_Init()` function to read the file and initialize the system. After initialization, the application need only call the `CDI_Convert()` function to obtain two arrays containing the wavelengths and amplitudes of the spectral data.

Installation

The DLL may be installed in any subdirectory, as long as the DLL's directory is in the PATH. The example files may also be installed in any directory. Simply run the `SETUP.EXE` on the distribution disk to copy the files into the working directory. The working directory will be added to the PATH statement by `SETUP`.

Example Program

An example program written in Microsoft Visual C/C++ is supplied with the DLL. The example source code shows how to use explicit dynamic linking to call each of the driver functions. The example also shows how to set up a timer loop using the time interval returned by the `CDI_Convert()` function.

Functions

Each function is described in the following section.

CDI Init

Syntax: `CDI_Init(char* SetupFile, float* WavelenArray, long* ArrayLength, short int InitFlashFileNum, short int BoardAddress)`

This function loads the initialization files stored by the CDI SPEC Windows application software. The setup file, with extension .STP, contains the instrument settings. Background and reference traces are stored in files of the same name as the .STP file, with different extensions. The .R\$\$ files are also required for initialization, and must reside in the same directory as the .STP file. It is recommended that setup files are created using the CDI SPEC application and stored in the \CDI directory to assure that all required files exist. `CDI_Init` may be called at any time to load a new setup file and re-initialize the instrument settings. If the specified on-board memory file read is unsuccessful, an attempt is made to initialize from the file `SetupFile`. The result is indicated by the return value.

Parameters

`SetupFile` Pointer to the null terminated string containing the full pathname for the setup file.
`WavelenArray` Pointer to the float array of X-values (wavelengths).
`ArrayLength` Pointer to the long variable in which the length of the data arrays is returned.
`InitFlashFileNum` short int variable which specifies whether to initialize from the setup file specified by `SetupFile` or from the on-board flash memory. If `InitFlashFileNum < 0`, then initialization is from the file `SetupFile`. There are 6 possible on-board memory files which can be specified using the following identifiers defined in `CDI32.H`:

InitFlashFileNum	Definition
<code>CAL_Factory1</code>	Factory Calibration File Number 1
<code>CAL_Factory2</code>	Factory Calibration File Number 2
<code>CAL_Factory3</code>	Factory Calibration File Number 3
<code>CAL_Factory4</code>	Factory Calibration File Number 4
<code>CAL_User1</code>	User Calibration File Number 1
<code>CAL_User2</code>	User Calibration File Number 2
<code>InitFlashFileNum < 0</code>	Initialize from <code>SetupFile</code> disk file

`BoardAddress` short int variable which specifies the I/O address occupied by the spectrograph card.

Return Value Returns is a short int greater than or equal to 0 for a successful initialization. If less than zero, a serious error has occurred, and none of the DLL functions should be called. The following identifiers are defined in `CDI32.H`:

<code>INIT_Failed</code>	Initialization failed.
<code>INIT_Illegal_Addr</code>	Initialization failed due to the board address being out of range.
<code>INIT_From_File</code>	Initialization from file <code>SetupFile</code> succeeded.
<code>INIT_From_Flash</code>	Initialization from the specified on-board memory file succeeded.

Comments This function performs the same action as the File Open Setup command of the CDI SPEC application, with the option of initializing from on-board memory.

Example

```
// Declare Global Variables
#define MAX_DATA 4097
HGLOBAL hX, hY;
float *lpY, *lpX;
long NumPts;
float CCDTemp, TimerInt;
```

```

char StatusString[80] = "";
HINSTANCE hDLLInst;
BOOL SampleDone = FALSE;
BOOL alreadyinit = FALSE;

void Call_Init()
{
typedef short int (WINAPI *LPCDECLFUNC)(char * SetupFile, float * WavelenArray,
    long * ArrayLength, short int InitFlashFileNum,
    short int BoardAddress );
LPCDECLFUNC CDI_Init = NULL; // Declare pointers to functions that can be type-checked.
short int i, nResult;

if(!alreadyinit)
{
hDLLInst = LoadLibrary ("CDI32.DLL");
if (hDLLInst == NULL)
{
MessageBox ("LoadLibrary Failed.", "DLL Error", MB_OK);
}

// allocate global data arrays
hY = GlobalAlloc (GHND, sizeof (float) * (MAX_DATA));
if (hY == 0) return;
lpY = (float *) GlobalLock(hY);
if (lpY == 0) return;
for (i = 0; i < MAX_DATA-1; i +=1)
    lpY[i] = (float).001;

hX = GlobalAlloc (GHND, sizeof (float) * (MAX_DATA));
if (hX == 0) return;
lpX = (float *) GlobalLock(hX);
if (lpX == 0) return;
for (i = 0; i < MAX_DATA-1; i +=1)
    lpX[i] = (float).001;
NumPts = (long)0;

alreadyinit = TRUE;
}

if (hDLLInst != NULL)
{
CDI_Init = (LPCDECLFUNC)GetProcAddress (hDLLInst, "CDI_Init");
if (CDI_Init)
    nResult = CDI_Init("c:\\cdi\\setup$1.s$$", lpX, &NumPts, -1, (int)BoardAddr);
}
} // end Call_Init()

```

CDI Convert

Syntax: CDI_Convert (float *SpecData, float *TempData, char *StatBar, float *Timer, BOOL* SampleDone, float* WavelenArray, long* ArrayLength, short int ForceRead)

This function samples the conversion complete status of the CCD array and FIFOs, and returns the fully processed spectral data when the conversion is complete. The next call to the function should be executed after the time interval returned in Timer.

Parameters

SpecData	Pointer to the float array of Y-values (amplitudes) of the spectral data.
TempData	Pointer to the float variable in which the CCD temperature sensor data is returned.
StatBar	Pointer to the char string in which the status string is returned. The status string contains the string displayed in the status bar of the CDI SPEC Windows application.
Timer	Pointer to the float variable in which the recommended time interval (in seconds) to the next call to CDI_Convert is returned. Timer is used to set the Windows timer for the next interval.
SampleDone	Pointer to the BOOL variable in which the conversion complete status is returned. SampleDone is TRUE when SpecData and IpWavelength are updated.
WavelenArray	Pointer to the float array of X-values (wavelengths).
ArrayLength	Pointer to the long variable in which the length of the data arrays is returned.
ForceRead	short int variable which when greater than zero (ForceRead >0) forces a FIFO read, overriding all timing logic internal to the driver functions. Set ForceRead > 0 if the Timer value is not used to time calls to CDI_Convert.
Return Value	Returns is greater than or equal to 0. If less than zero, a serious error has occurred.

Example:

```
// Declare Global Variables
#define MAX_DATA 4097
HGLOBAL hX, hY;
float *IpY, *IpX;
long NumPts;
realtyp CCDTemp, TimerInt;
char StatusString[80] = "";
HINSTANCE hDLLInst;
BOOL SampleDone = FALSE;

// CDI_Convert can be called from a Windows timer CALLBACK function
extern "C" UINT __export CALLBACK ConvertTimer (HWND hWnd, UINT nMsg, UINT
nIDEvent, DWORD dwTime)
{
typedef int (FAR PASCAL *LPCDECLFUNC)(float *SpecData, float *TempData, char *StatBar,
float *Timer, BOOL* SampleDone, float* IpWavelength, long* ArrayLength);
int nResult = 0;
static BOOL alreadyinit = FALSE;
LPCDECLFUNC CDI_Convert = NULL;

KillTimer (hWnd, IDT_CONVERT_TIMER);
if (hDLLInst >= HINSTANCE_ERROR)
{
```

```

CDI_Convert = (LPCDECLFUNC)GetProcAddress (hDLLInst, "_CDI_Convert");
if (CDI_Convert)
    nResult = CDI_Convert(lpY, &CCDTemp, StatusString, &TimerInt,
        &SampleDone, lpX, &NumPts);
}
if(!SetTimer (hWnd, IDT_CONVERT_TIMER, (UINT)( (long)(TimerInt * 1000.0) ),
    lpfnConvertTimer))
    MessageBox (hWnd, "Too many clocks or timers!", "ProcessCommands",
        MB_TASKMODAL | MB_OK | MB_ICONEXCLAMATION);
return 0;
}

```

CDI TimerEnable

Syntax: CDI_TimerEnable(short int TimerEnable, int *HandShake, HWND HwndParent)

This function enables a self-running timer loop implemented with a self-contained callback function and WM_TIMER messages. When the timer is enabled, the calling program may call CDI_UpdateData() at any time to read the most recent fully processed spectral data.

Parameters

TimerEnable	short integer variable which when greater than zero (TimerEnable > 0) enables the DLL's internal timer callback function. When not greater than zero (TimerEnable <= 0) the DLL's internal timer callback function is disabled.
HandShake	Pointer to the integer variable which indicates the number of samples completed by the timer routine. HandShake may be reset to zero by the calling application and polled as a ready flag.
HwndParent	Parent window handle. If NULL, the timer window is created with WS_POPUP style. If not NULL, then the timer window is created with WS_CHILD style.
Return Value	No return value.

CDI UpdateData

Syntax: CDI_UpdateData(float* WavelenArray, float *SpecData, float *TempData, long* ArrayLength)

This function returns the most recent fully processed spectral data.

Parameters

WavelenArray	Pointer to the float array of X-values (wavelengths).
SpecData	Pointer to the float array of Y-values (amplitudes) of the spectral data.
TempData	Pointer to the float variable in which the CCD temperature sensor data is returned.
ArrayLength	Pointer to the long variable in which the length of the data arrays is returned.
Return Value	No return value.

CDI SetShowoffFlag

Syntax: CDI_SetShowoffFlag(BOOL ShowOff, float *counts)

This function is called prior to calling CDI_ChangeOffset to obtain the current offset command, since the new analog offset will be relative to the existing command. The next spectrum read after the ShowOff

flag is set to TRUE is in raw A/D counts with no offset subtracted, and this mode will continue until the ShowOff flag is set to FALSE.

Parameters

ShowOff	Pointer to the BOOL variable in which toggles the show offset command. If ShowOff is TRUE then the amplitude calibration mode is overridden so that the raw counts can be returned by CDI_Convert on the next call. When ShowOff is set to FALSE then the amplitude calibration mode is restored.
counts	Pointer to the float variable in which the current analog offset DAC command is returned.
Return Value	No return value.
Comments	This function when combined with CDI_ChangeOffset performs the same action as the Setup Analog Offset Countscommand of the CDI SPEC application.

CDI_ChangeOffset

Syntax: CDI_ChangeOffset(float *newCounts)

This function is called to change the analog offset counts commanded to the offset DAC.

Parameters

newCounts	Pointer to the float variable which contains the analog offset counts command.
Return Value	No return value.

CDI_ChangeIntTime

Syntax: CDI_ChangeIntTime(float *newIntTime, float *Timer)

This function is called to set the CCD integration time or exposure time.

Parameters

newIntTime	Pointer to the float variable which contains the integration time in seconds.
Timer	Pointer to the float variable in which the recommended time interval (in seconds) to the next call to CDI_Convert is returned. Timer may be used to set the Windows timer for the next interval.
Return Value	No return value.
Comments	This function performs the same action as the Setup Integration Time command of the CDI SPEC application.

CDI_SetCCDTemp

Syntax: CDI_SetCCDTemp(float* m_TempCmd, BOOL m_On)

This function is called to set the thermoelectric cooler (TEC) operating temperature.

Parameters

m_TempCmd	Pointer to the float variable which specifies the setpoint temperature for the thermoelectric cooler.
m_On	BOOL variable which when TRUE turns on the thermoelectric cooler.
Return Value	No return value.

Comments This function performs the same action as the Setup Temperature Control command of the CDI SPEC application.

CDI ChangeCompensation

Syntax: CDI_ChangeCompensation(float* m_BBTemp, float* m_Multiplier, short int* CMode, short int* BMode, short int* RMode, char* m_RefFilText, char* m_BinRefFile)

This function is called to select amplitude calibration units and background subtraction.

Parameters

m_BBTemp	Pointer to the float variable which specifies the ideal blackbody color temperature for computation of the blackbody calibration.
m_Multiplier	Pointer to the float variable which specifies the scale factor which multiplies the normalized blackbody calibration. m_Multiplier specifies the maximum value of the output.
CMode	Pointer to the short integer variable which specifies the amplitude calibration mode. CMode identifiers (include CDI32.H): CM_ZeroOffset = Zero Offset Comp CM_SubBgnd = Subtract Background CM_NormMax = Percent Transmittance CM_NormAU = Absorbance Units (log (1/T)) CM_NormDB = Normalize (dB) CM_NormBB = Blackbody Calibration CM_NormFile = File Calibration using file named in m_RefFilText
BMode	Pointer to the short integer variable which specifies the background subtraction mode. BMode identifiers (include CDI32.H): BM_InternalBG = Internal background trace BM_CCDDarkPix = Live CCD dark sensor pixels BM_IntBGwTracking = Combine internal background and live CCD dark sensor pixels
RMode	Pointer to the short integer variable which specifies the reference trace to be used for amplitude calibration. RMode identifiers (include CDI32.H): RM_InternalRef = Internal reference trace RM_OnboardRef = Default reference stored in file default.ref RM_BinRefFile = Loads a binary reference file named in m_BinRefFile
m_RefFilText	char* to string which contains the name of the file containing calibration data when CMode = CM_NormFile
m_BinRefFile	char* to string which contains the name of the file containing the reference when RMode = RM_BinRefFile.
Return Value	No return value.
Comments	This function performs the same action as the Setup Amplitude Calibration command of the CDI SPEC application.

CDI ChangeTriggerMode

Syntax: CDI_ChangeTriggerMode(unsigned short int *TrigMode)

This function is called to change the trigger mode of the system.

Parameters

TrigMode	Pointer to the unsigned short integer variable which specifies the trigger mode. TrigMode codes:
TM_SynchRead	= Internal Trigger, Read Single Spectrum on each sweep
TM_AvgBuffer	= Internal Trigger, Read and Average 17 Spectra on each sweep
TM_IntPixRst	= Internal Trigger, Quick Change Integration Time and High Amplitude Stability
TM_IntFast	= Internal Trigger, Minimum Integration Time and Trigger Latency
TM_ExtFast	= External Trigger, Minimum Integration Time and Trigger Latency
TM_ExtPixRst	= External Trigger, High Amplitude Stability
TM_MultiBegin	= External Trigger at Beginning of Event, Read and Display 18 Spectra
TM_MultiMid	= External Trigger at Middle of Event, Read and Display 18 Spectra
TM_MultiEnd	= External Trigger at End of Event, Read and Display 18 Spectra
TM_MultiBeginAvg	= External Trigger at Beginning of Event, Read and Average 18 Spectra on each sweep
TM_MultiMidAvg	= External Trigger at Middle of Event, Read and Average 18 Spectra on each sweep
TM_MultiEndAvg	= External Trigger at End of Event, Read and Average 18 Spectra on each sweep
Return Value	No return value.
Comments	This function performs the same action as the Trigger Mode command of the CDI SPEC application.

CDI_ChangeBoardAddr

Syntax: CDI_ChangeBoardAddr(short int *Addr, float *Timer)

This function is called to change the I/O address of the spectrograph card.

Parameters

Addr	Pointer to the short int variable specifying the I/O address which matches the switch settings on the board. Legal values are between 512 and 1008, inclusive.
Timer	Pointer to the float variable in which the recommended time interval (in seconds) to the next call to CDI_Convert is returned. Timer is used to set the Windows timer for the next interval. This function issues a Start Convert signal.
Return Value	No return value.
Comments	This function performs the same action as the File New Setup Board Address command of the CDI SPEC application.

CDI_GetIntegrateTime

Syntax: CDI_GetIntegrateTime(float *IntTime, float *Timer);

This function is called to get the current CCD integration time (exposure time) from the DLL driver library.

Parameters

IntTime	Pointer to the float variable in which the current integration time in seconds is returned.
Timer	Pointer to the float variable in which the recommended time interval (in seconds) to the next call to CDI_Convert is returned. Timer may be used to set the Windows timer for the next interval.
Return Value	No return value.

CDI_GetTriggerMode

Syntax: `CDI_GetTriggerMode(unsigned short int *TrigMode)`

This function is called to get the current trigger mode of the system.

Parameters

TrigMode Pointer to the unsigned short integer variable in which the current trigger mode is returned.

TrigMode codes:

TM_SynchRead = Internal Trigger, Read Single Spectrum on each sweep
TM_AvgBuffer = Internal Trigger, Read and Average 17 Spectra on each sweep
TM_IntPixRst = Internal Trigger, Quick Change Integration Time and High Amplitude Stability
TM_IntFast = Internal Trigger, Minimum Integration Time and Trigger Latency
TM_ExtFast = External Trigger, Minimum Integration Time and Trigger Latency
TM_ExtPixRst = External Trigger, High Amplitude Stability
TM_MultiBegin = External Trigger at Beginning of Event, Read and Display 18 Spectra
TM_MultiMid = External Trigger at Middle of Event, Read and Display 18 Spectra
TM_MultiEnd = External Trigger at End of Event, Read and Display 18 Spectra
TM_MultiBeginAvg = External Trigger at Beginning of Event, Read and Average 18 Spectra on each sweep
TM_MultiMidAvg = External Trigger at Middle of Event, Read and Average 18 Spectra on each sweep
TM_MultiEndAvg = External Trigger at End of Event, Read and Average 18 Spectra on each sweep
Return Value No return value.

CDI_GetCompensation

Syntax: `CDI_GetCompensation(float* m_BBTemp, float* m_Multiplier, short int* CMode, short int* BMode, short int* RMode, char* m_RefFilText, char* m_BinRefFile)`

This function is called to get the current amplitude calibration mode.

Parameters

m_BBTemp pointer to a float variable in which the current value of the blackbody color temperature is returned.

m_Multiplier pointer to a float variable in which the current value of the scale factor which multiplies the normalized blackbody calibration is returned.

CMode pointer to an short integer variable in which the current value of the calibration mode is returned. CMode identifiers (include CDI32.H):

CM_ZeroOffset = Zero Offset Comp
CM_SubBgnd = Subtract Background
CM_NormMax = Percent Transmittance
CM_NormAU = Absorbance Units (log (1/T))
CM_NormDB = Normalize (dB)
CM_NormBB = Blackbody Calibration
CM_NormFile = File Calibration using file named in m_RefFilText

BMode pointer to an short integer variable in which the current value of the background subtraction mode is returned. BMode identifiers (include CDI32.H):

BM_InternalBG = Internal background trace
BM_CCDDarkPix = Live CCD dark sensor pixels

	BM_IntBGwTracking = Combine internal background and live CCD dark sensor pixels
RMode	pointer to a short integer variable in which the current value of the reference mode is returned. RMode identifiers (include CDI32.H): RM_InternalRef = Internal reference trace RM_OnboardRef = Default reference stored in file default.ref RM_BinRefFile = Loads a binary reference file named in m_BinRefFile
m_ReffilText	char* to string which contains the name of the file containing calibration data when CMode = CM_NormFile
m_BinRefFile	char* to string which contains the name of the file containing the reference when RMode = RM_BinRefFile.
Return Value	No return value.

CDI LinearizeLive

Syntax: CDI_LinearizeLive(short int m_Linearize, float* m_Xstep, float* m_Xstop, float* m_Xstart, BOOL m_AllTraces)

This function is called to select the linear interpolation mode to provide equally spaced data points.

Parameters

m_Linearize	BOOL variable which when TRUE enables linear interpolation of realtime spectral data.
m_Xstep	Pointer to the float variable which specifies the point spacing in nanometers.
m_Xstop	Pointer to the float variable which specifies the end wavelength in nanometers of the linearized data.
m_Xstop	Pointer to the float variable which specifies the end wavelength in nanometers of the linearized data.
m_AllTraces	BOOL variable which when TRUE enables linear interpolation of all traces
Return Value	No return value.
Comments	This function performs the same action as the Setup Linearize command of the CDI SPEC application.

CDI SetupDualbeam

Syntax: CDI_SetupDualbeam(BOOL EnableDualBeamMode)

This function is called to enable dual beam operation on specially configured dual beam spectrograph systems.

Parameters

EnableDualBeamMode	BOOL variable which when TRUE enables the dual beam reference computation.
Return Value	Returns negative value if not successful.
Comments	This function performs the same action as the Setup Dual Beam Mode command of the CDI SPEC application.

CDI OnStoreCopylive

Syntax: CDI_OnStoreCopylive(BOOL m_Background, BOOL m_ReCalc)

This function is called to capture the current sample and store it internally for use as the light reference or dark background in the amplitude calibration calculations.

Parameters

m_Background	Pointer to the BOOL variable which selects the background store function. If m_Background is TRUE, then the current trace is stored to the internal background array. If m_Background is FALSE, then the current trace is stored to the internal reference array.
m_ReCalc	Pointer to the BOOL variable which enables recalculation of the compensated reference. m_ReCalc should be TRUE when storing the reference, and FALSE when storing the background, unless the background is stored after the reference, then m_ReCalc should be TRUE when storing the background .
Return Value	No return value.
Comments	This function performs the same action as the Trace Store Internal command of the CDI SPEC application.

CDI_OnDFStoreCopylive

Syntax: CDI_OnDFStoreCopylive(BOOL m_DarkLevel, BOOL m_ReCalc, BOOL m_DoubleFiber, BOOL m_RefChanRef, BOOL m_SignalChanRef, BOOL m_CalcRatio)

This function is used only for double input fiber units. The function is similar to CDI_OnStoreCopylive(), with the added capability to select the reference or signal channel.

Parameters

m_DarkLevel	BOOL variable which selects the background store function. If m_Background is TRUE, then the current trace is stored to the internal background array. If m_Background is FALSE, then the current trace is stored to the internal reference array specified by the parameters m_RefChanRef and m_SignalChanRef.
m_ReCalc	BOOL variable which enables recalculation of the compensated reference. m_ReCalc should be TRUE when storing the reference, and FALSE when storing the background, unless the background is stored after the reference, then m_ReCalc should be TRUE when storing the background. When m_ReCalc is TRUE when storing the background, then the reference channel reference, the signal channel reference, and the ratio of the reference channel to the signal channel are all updated.
m_DoubleFiber	BOOL variable which when set to TRUE enables the double fiber functions. When FALSE, the function is identical to OnStoreCopylive().
m_RefChanRef	BOOL variable which, when TRUE, stores the reference for the reference channel.
m_SignalChanRef	BOOL variable which, when TRUE, stores the reference for the signal channel.
m_CalcRatio	BOOL variable which enables recalculation of the ratio of the reference channel to the signal channel. This ratio is used to compensate the signal channel whenever the reference channel is updated, similar to a dual beam reference mode.
Return Value	No return value.
Comments	This function performs the same action as the Trace Store Internal command of the CDI SPEC application.

CDI Rereference

Syntax: CDI_Rereference(void)

This function is called to acquire and store new background and reference traces. A message box is displayed to prompt the user to connect dark and reference inputs at the appropriate times. For units specially equipped with fiber optic switches, no prompts are provided and the switch is automatically actuated to select dark and reference inputs.

Return Value	No return value.
Comments	This function performs the same action as the Setup Acquire Background and Reference command of the CDI SPEC application.

CDI Acqbg

Syntax: CDI_Acqbg(void)

This function is called to acquire and store a new background trace. A message box is displayed to prompt the user to connect the dark input at the appropriate time. For units specially equipped with fiber optic switches, no prompts are provided and the switch is automatically actuated to select and de-select the dark input.

Return Value	No return value.
Comments	This function performs the same action as the Setup Acquire Background command of the CDI SPEC application.

CDI Acqref

Syntax: CDI_Acqref(void)

This function is called to acquire and store a new reference trace. A message box is displayed to prompt the user to connect the reference input at the appropriate time. For units specially equipped with fiber optic switches, no prompts are provided and the switch is automatically actuated to select and de-select the reference input.

Return Value	No return value.
Comments	This function performs the same action as the Setup Acquire Reference command of the CDI SPEC application.

CDI SaveSetupFile

Syntax: CDI_SaveSetupFile(char *FileName)

This function is called to save the current instrument and processing settings to a file. The file can later be loaded to initialize the system using CDI_Init.

Parameters

m_FileName	char* to the string which contains the name of the file in which the current setup is to be stored.
Return Value	No return value.
Comments	This function performs the same action as the File Save Setup command of the CDI SPEC application.

CDI LoadEESetup

Syntax: CDI_LoadEESetup(short int m_Calfile)

This function is called to load instrument and processing settings from the setup data conained in the on-board flash memory.

Parameters

m_Calfile short int variable which selects the on-board memory file to be loaded. The following identifiers are defined in CDI32.H:

<u>m_Calfile</u>	<u>Definition</u>
CAL_Factory1	Factory Calibration File Number 1
CAL_Factory2	Factory Calibration File Number 2
CAL_Factory3	Factory Calibration File Number 3
CAL_Factory4	Factory Calibration File Number 4
CAL_User1	User Calibration File Number 1
CAL_User2	User Calibration File Number 2

Return Value Return value is greater than zero (positive) if the load is successful. If unsuccessful, a zero or negative value is returned. In the event of a failure, it is recommended that a default setup file is loaded from disk.

Comments This function performs the same action as the File Load Setup from Flash command of the CDI SPEC application.

CDI SaveEESetup

Syntax: CDI_SaveEESetup(short int m_Calfile)

This function is called to save instrument and processing settings to the on-board flash memory.

Parameters

m_Calfile short int variable which selects the on-board memory file to be written. The following identifiers are defined in CDI32.H:

<u>m_Calfile</u>	<u>Definition</u>
CAL_Factory1	Factory Calibration File Number 1
CAL_Factory2	Factory Calibration File Number 2
CAL_Factory3	Factory Calibration File Number 3
CAL_Factory4	Factory Calibration File Number 4
CAL_User1	User Calibration File Number 1
CAL_User2	User Calibration File Number 2

Return Value Return value is greater than zero (positive) if the save is successful. If unsuccessful, a zero or negative value is returned.

Comments This function performs the same action as the File Save Setup to Flash command of the CDI SPEC application.

CDI_SetupColorimetry

Syntax: CDI_SetupColorimetry(short int m_OnLineColor, short int m_CIE_Observer, double m_RefL_star, double m_Refa_star, double m_Refb_star)

This function is called to setup the colorimetry calculations. Colorimetry data is returned in the COLOR_TYPE structure as defined in CDI32.H.

Parameters

m_OnLineColor short int variable which when greater than 0 enables the realtime colorimetry calculations
m_CIE_Observer short int variable which when 0 selects the CIE 1931 2 deg observer and when 1 selects the CIE 1964 10 deg observer
m_RefL_star double variable which specifies the L* value to be used as the reference in the delta E* (ab) color difference equation.
m_Refa_star double variable which specifies the a* value to be used as the reference in the delta E* (ab) color difference equation.
m_Refb_star double variable which specifies the b* value to be used as the reference in the delta E* (ab) color difference equation.

Return Value No return value.

Comments This function performs the same action as the Setup Colorimetry command of the CDI SPEC application.

CDI_GetColorimetry

Syntax: CDI_GetColorimetry(COLOR_TYPE *Colorimetry)

Parameters

Colorimetry Pointer to the COLOR_TYPE structure in which all colorimetry data is returned. The COLOR_TYPE structure is defined in CDI32.H

Return Value No return value.

CDI_SetupAveraging

Syntax: CDI_SetupAveraging(float m_Samples)

This function is called to setup the number of samples to be averaged. The current sample counter is not reset in order to allow extending the average beyond the originally specified value. To reset the sample counter (and reset the average) call CDI_ResetAverage after CDI_SetupAveraging.

Parameters

m_Samples float variable which specifies the number of samples to be accumulated and averaged. If m_Samples=1, averaging is OFF.

Return Value No return value.

Comments This function performs the same action as the Avg Sample edit control in the Control Bar of the CDI SPEC application.

CDI_ResetAverage

Syntax: CDI_ResetAverage(void)

This function is called to reset the current sample counter, resetting the accumulated average.

Return Value	No return value.
Comments	This function performs the same action as the Setup Online Processing Reset Average command of the CDI SPEC application.

CDI_GetAvgSamples

Syntax: CDI_GetAvgSamples(void)

This function is called to read the total number of samples to be accumulated in the average.

Return Value	Returns the short integer value of the number of samples to be accumulated and averaged.
--------------	--

CDI_GetAvgN

Syntax: CDI_GetAvgN(void)

This function is called to read the current sample counter which indicates the current number of samples accumulated in the average.

Return Value	Returns the short integer value of the current sample counter (number of samples accumulated and averaged).
--------------	---

CDI_GetNumPtsLive

Syntax: CDI_GetNumPtsLive(long *NumPts)

Parameters

NumPts	Pointer to the long integer variable in which the number of points in the spectral data arrays (wavelengths and amplitudes) is returned.
--------	--

Return Value	No return value.
--------------	------------------

CDI_SaveDataFile

Syntax: CDI_SaveDataFile(char *FileName, char *FileComment,
 BOOL m_InstrHdr, BOOL m_ColorHdr, BOOL m_TraceData)

This function is called to save the current trace to an ASCII data file, with the option of storing an ASCII header with or without the data.

Parameters

m_FileName	char* to the string which contains the name of the file in which the current setup is to be stored.
------------	---

m_FileComment	char* to the string which contains an optional comment string to be included in the ASCII header, if selected.
m_InstrHdr	BOOL variable which selects the instrument settings header data to be included in the data file.
m_ColorHdr	BOOL variable which selects the colorimetry header data to be included in the data file.
m_TraceData	BOOL variable which selects the trace, or spectral data, to be included in the data file. If no headers are selected, the m_TraceData defaults to TRUE.
Return Value	No return value.
Comments	This function performs the same action as the File Save Trace command of the CDI SPEC application.

CDI TriggerBurst

Syntax: CDI_TriggerBurst(unsigned char *ram_buffer, long max_buffer_size, short int max_samples, long *samples_returned, long *floats_per_sample)

This function is called to start the acquisition of a temporal sample. The function waits for the the external trigger then starts reading data. The function starts reading data immediately if internal trigger mode is selected. The spectra are processed, then returned in the memory object ram_buffer.

Parameters

ram_buffer	unsigned char __huge* to the memory object in which the data is returned.
max_buffer_size	long variable which contains the size of ram_buffer in bytes.
max_samples	short int variable which specifies the number of spectra to acquire.
samples_returned	long * to the variable in which the number of samples acquired is returned.
floats_per_sample	long * to the variable in which the number of floating point values per spectrum is returned.

Return Value A short int is returned which indicates the number of spectra acquired. If 0 is returned, then an error has occurred.

Comments This function performs the same action as the Trigger Start Temporal Sample command of the CDI SPEC application.

CDI SetupBurstMode

Syntax: CDI_SetupBurstMode(void)

This function is called to select the temporal sampling mode prior the the start of a temporal sample.

Comments This function performs the same action as the Trigger Select Temporal Sample Mode (same as Trigger Mode Multiple Trace Capture Mode (external trigger) Trigger Position - Beginning) command of the CDI SPEC application.

CDI GetDualbeamMode

Syntax: CDI_GetDualbeamMode(short int* pDualBeamMode)

This function is called to determine whether the dual beam reference operating mode is selected.

Parameters

pDualBeamMode pointer to a short int variable in which the current reference mode is returned:

If Dual Beam Reference Operating Mode is ACTIVE, then pDualBeamMode = 1,
else pDualBeamMode = 0 (single beam operating mode).

Return Value No return value.

CDI GetRefBgStatus

Syntax: CDI_GetRefBgStatus(float* m_RMean, float* m_RDelta, short int* m_RDone, float* m_BMean,
float* m_BDelta, short int* m_BDone)

This function is called to interrogate the DLL after commanding the acquisition of a new reference or background to determine whether the operation has completed, and to obtain statistics.

Parameters

m_RMean float * to the variable in which the reference noise mean is returned.
m_RDelta float * to the variable in which the reference mean delta is returned.
m_RDone short int * to the variable in which the reference complete status is returned.
 m_RDone = 0 for not done, m_RDone = 1 for done.
m_BMean float * to the variable in which the background noise mean is returned.
m_BDelta float * to the variable in which the background mean delta is returned.
m_BDone short int * to the variable in which the reference complete status is returned.
 m_BDone = 0 for not done, m_BDone = 1 for done.

CDI ShowMessages

Syntax: CDI_ShowMessages(short int m_ShowMessages)

This function is called to enable or disable the message boxes which prompt the user and announce when the operation is complete.

Parameters

m_ShowMessages short int variable in which the enable switch is passed to the DLL1.
 m_ShowMessages = 0 to disable messages,
 m_ShowMessages = 1 to enable messages.

CDI OpticSwitch

Syntax: CDI_OpticSwitch(BOOL Switch)

This function is called to control the shutter of the Stablamp light source, which is wired to the discrete output Optic Switch 1.

Parameters

Switch BOOL variable in which the switch command is passed.
 Switch = FALSE to close the shutter.
 Switch = TRUE to open the shutter.

CDI OpticSwitch2

Syntax: CDI_OpticSwitch2(BOOL Switch)

This function is called to control the discrete output Optic Switch 2.

Parameters

Switch BOOL variable in which the switch command is passed.
Switch = FALSE for OFF (inactive state)
Switch = TRUE for ON(active state)

CDI_ReadUnitSerNo

Syntax: CDI_ReadUnitSerNo(char *pSerialNo, char *pEPLDRev, int *pFifoSizeK, int *pDetectorType)

This function is called to interrogate a spectrograph unit to identify its serial number and hardware configuration. The function CDI_ChangeBoardAddr() should be called to select the board address to be interrogated by this function.

Parameters

pSerialNo pointer to the char variable in which the unit serial number string is returned.
pEPLDRev pointer to the char variable in which the EPLD revision string is returned.
pFifoSizeK pointer to the int variable in which the size of the on-board FIFOs is returned. The size is in kWords; e.g. the size is (pFifoSizeK * 1024) words.
pDetectorType pointer to the int variable in which the type of detector is returned. The type of detector indicates the model of the spectrometer.

CDI_ChangeGain

Syntax: CDI_ChangeGain(BOOL HighGain)

This function is called to change the detector gain (currently only available only on InGaAs NIR units).

Parameters

HighGain BOOL variable in which the command is passed.
HighGain = FALSE to select low gain.
HighGain = TRUE to select high gain

CDI_InputPixelData

Syntax: CDI_InputPixelData(float *SpecData, long * ArrayLength)

This function is called to load the internal data arrays of the DLL with user-defined pixel data. The DLL functions can then be used to calculate using the externally generated data.

Parameters

SpecData pointer to the float variable in which the input pixel data is passed.
ArrayLength pointer to the long int variable in which the length of the input pixel data array is passed (the number of floats).

CDI_OutputPixelData

Syntax: CDI_OutputPixelData(float *SpecData, long * ArrayLength)

This function is called to read raw pixel data from the DLL for test purposes.

Parameters

SpecData	pointer to the float variable in which the output pixel data is returned.
ArrayLength	pointer to the long int variable in which the length of the output pixel data array is returned (the number of floats).

CDI_ReadData

Syntax: CDI_ReadData(float * lpWavelength, float *SpecData, long * ArrayLength)

This function is called to compute the spectral data, including spectral units, peaks, color, and all other calculations. Provided for test purposes.

Parameters

lpWavelength	pointer to the float variable in which the output wavelength data is returned.
SpecData	pointer to the float variable in which the output spectral data is returned.
ArrayLength	pointer to the long int variable in which the length of the output pixel data array is returned (the number of floats).

CDI_ChangeNoBoard

Syntax: `CDI_ChangeNoBoard(BOOL bNoBoard)`

This function is called with `bNoBoard = TRUE` to disable commands to the hardware. Use this function when calling `ReadData()` for test purposes.

Parameters

`bNoBoard` `BOOL` variable in which the command is passed.
 `bNoBoard = FALSE` for normal operation with hardware.
 `bNoBoard = TRUE` to disable commands to hardware.

CDI_SetupColorimetryEx

Syntax: `CDI_SetupColorimetryEx(short int m_OnLineColor, short int m_CIE_Observer, float * m_RefL_star, float * m_Refa_star, float * m_Refb_star, short int m_Ref_Illum, float * m_ref_illum_degK, short int m_Online_CRI, short int m_CRI_RefIll, float * m_CRI_RefDegK, float * SpecDataIn, long PtsIn, short int m_Update_Color, short int m_Update_CRI)`

This function is called to setup the colorimetry calculations. Colorimetry data is returned in the `COLOR_TYPE` structure as defined in `CDI32.H`.

Parameters

`m_OnLineColor` short int variable which when greater than 0 enables the realtime colorimetry calculations

`m_CIE_Observer` short int variable which when 0 selects the CIE 1931 2 deg observer and when 1 selects the CIE 1964 10 deg observer

`m_RefL_star` double variable which specifies the L* value to be used as the reference in the delta E* (ab) color difference equation.

`m_Refa_star` double variable which specifies the a* value to be used as the reference in the delta E* (ab) color difference equation.

`m_Refb_star` double variable which specifies the b* value to be used as the reference in the delta E* (ab) color difference equation.

`m_Ref_Illum` pointer to short int variable which specifies the CIE standard reference illuminant for the color coordinate calculation, eg. `D_65`. See `typ_Illum` definition in `CDI32.H`.

`m_ref_illum_degK` pointer to float variable which specifies the color temperature in degrees Kelvin of the "other" reference illuminant (selected by `m_Ref_Illum = RI_Other`).
`m_ref_illum_degK` must be < 25,000 deg K.

`m_Online_CRI` pointer to short int variable which when greater than 0 enables the realtime calculation of the color redering index.

`m_CRI_RefIll` pointer to short int variable which specifies the Nearest CIE standard reference illuminant for the color renering index calculation. See `typ_Illum` definition in `CDI32.H`.

`m_CRI_RefDegK` pointer to float variable which specifies the color temperature in degrees Kelvin of the "other" reference illuminant (selected by `m_CRI_RefIll = RI_Other`).
`m_CRI_RefDegK` must be < 25,000 deg K.

`SpecDataIn` pointer to floating point array which holds the spectral data input to the colorimetry and color rendering index calculation. If `NULL`, the most recent DLL internal spectral data is used.

`PtsIn` pointer to the long int variable in which the length of the `SpecDataIn` data array is specified (the number of floats).

m_Update_Color	short int variable which when greater than 0 causes the recalculation of the colorimetry values independent of m_OnlineColor. Updated colorimetry values can then be obtained in a call to CDI_GetColorimetryEx()
m_Update_CRI	short int variable which when greater than 0 causes the recalculation of the color rendering index values independent of m_Online_CRI. Updated color rendering index values can then be obtained in a call to CDI_GetColorimetryEx()
Return Value	No return value.
Comments	This function performs the same action as the Setup Colorimetry and Setup Color Render Index commands of the CDI SPEC application.

CDI_GetColorimetryEx

Syntax: CDI_GetColorimetryEx(COLOR_TYPE * Colorimetry, short int m_OnLineColor, short int m_CIE_Observer, short int m_Online_CRI, short int m_CRI_RefIll, float * m_CRI_RefDegK, float * CRlarray9)

Parameters

Colorimetry	Pointer to the COLOR_TYPE structure in which all colorimetry data is returned. The COLOR_TYPE structure is defined in CDI32.H
m_OnLineColor	short int variable which when greater than 0 enables the realtime colorimetry calculations
m_CIE_Observer	short int variable which when 0 selects the CIE 1931 2 deg observer and when 1 selects the CIE 1964 10 deg observer
m_Online_CRI	pointer to short int variable which when greater than 0 enables the realtime calculation of the color rendering index.
m_CRI_RefIll	pointer to short int variable which specifies the Nearest CIE standard reference illuminant for the color rendering index calculation. See typ_Illum definition in CDI32.H.
m_CRI_RefDegK	pointer to float variable which specifies the color temperature in degrees Kelvin of the "other" reference illuminant (selected by m_CRI_RefIll = RI_Other). m_CRI_RefDegK must be < 25,000 deg K.
CRlarray9	pointer to an array of 9 floating point values, in which the color rendering indices are returned. Elements 0 through 7 return the individual CRI values. Element 8 returns the average of elements 0-7. If NULL, no values returned.

Return Value No return value.

CDI_GetPeakData

Syntax: CDI_GetPeakData(ARRAY_OF_PEAK_TYPE *PeaksDataOut)

This function is called to retrieve the peak detection data from the DLL. Peak data is returned in the ARRAY_OF_PEAK_TYPE and PEAK_TYPE structures as defined in CDI32.H.

Parameters

PeaksDataOut	Pointer to an ARRAY_OF_PEAK_TYPE structure in which the peak detection data is returned.
--------------	--

Return Value An integer value which indicates the number of peaks detected.

CDI_SetupPeakData

Syntax: `CDI_SetupPeakData(int BaseType,
float BaseThreshold,
float BasePctThreshold,
float PeakThreshold,
float DyDxMin,
BOOL m_PksOnline)`

This function is called to setup the peak detection calculations. Peak data is returned in the `ARRAY_OF_PEAK_TYPE` and `PEAK_TYPE` structures as defined in `CDI32.H`.

```
__declspec(dllexport) void WINAPI CDI_SetupPeakData(  
int BaseType,  
float BaseThreshold,  
float BasePctThreshold,  
float PeakThreshold,  
float DyDxMin,  
BOOL m_PksOnline);
```

Parameters

- BaseType** int variable in which the type of baseline is specified. The baseline is used to find the base (or start) of a peak. Three different baseline algorithms are defined:
BaseType = 0 to Select Dy/Dx (derivative) type Baseline
BaseType = 1 to Select Percent-of-Max type Baseline
BaseType = 2 to Select Fixed Level Baseline
- BaseThreshold** single precision float variable in which baseline threshold is specified in the currently selected spectral data units. The baseline threshold is used to identify potential peaks in the spectrum. The start of the peak occurs at the wavelength where the spectral amplitude exceeds the baseline threshold, and the end of the peak occurs at the wavelength where the spectral amplitude returns to a value less than the baseline threshold.
- BasePctThreshold** single precision float variable in which baseline threshold is specified in percent of maximum. BasePctThreshold is only used when a Percent-of-Max type Baseline selected, otherwise its value is ignored.
- PeakThreshold** single precision float variable in which peak threshold is specified in the currently selected spectral data units. The peak threshold is the minimum size peak which is detected by the algorithm. Whenever the maximum spectral amplitude between the baseline wavelengths exceeds the peak threshold, a peak is detected. Set PeakThreshold to the minimum size peak desired. If PeakThreshold is too small, then noise peaks may be detected.
- DyDxMin** single precision float variable in which derivative threshold is specified the currently selected spectral data units per nm. The derivative method computes the derivative of the spectrum (dY / dX), and finds the peaks near the zero crossing of the derivative. The max value of the spectrum is reported as a peak if it exceeds the Peak Threshold setting. This is the most sensitive method for peak detection, detecting small peaks even on the shoulders of larger peaks. Use the derivative method for sharp peaks such as emission lines. DyDxMin is used only when BaseType = 0 (derivative).

m_PksOnline	BOOL variable which enables (when TRUE) or disables (when FALSE) the automatic computation of peak detection for every spectrum acquired by the DLL. When TRUE, calling CDI_GetPeakData() will fetch the updated peak detection data.
Return Value	No return value.
Comments	This function performs the same action as the Setup Peak Detection command of the CDI SPEC application.

The special structures used to return the peak detection data are described below.

The maximum number of peaks detected by the DLL is MAX_PEAKS = 150. Each member of the PEAK_TYPE structure is described below.

```
typedef struct
{
    realtype    WaveLength; // The wavelength of the peak
    realtype    PeakValue;  // The amplitude of the peak
    realtype    BaseWav1;   // The wavelength of the start of the peak
    realtype    BaseAmp1;   // The amplitude at the wavelength BaseWav1
    realtype    BaseWav2;   // The wavelength of the end of the peak
    realtype    BaseAmp2;   // The amplitude at the wavelength BaseWav2
    realtype    FWHM;       // The peak full width at half of maximum
    realtype    FWHMWav1;   // The start wavelength at half of maximum
    realtype    FWHMAmp1;   // The amplitude at the wavelength FWHMWav1
    realtype    FWHMWav2;   // The end wavelength at half of maximum
    realtype    FWHMAmp2;   // The amplitude at the wavelength FWHMWav2
    realtype    FWHMCtrWav; // The wavelength at the center of FWHMWav1 and
FWHMWav2
    realtype    FWHMCtrAmp; // The amplitude at the wavelength FWHMCtrWav
    realtype    FWHMArea;   // The area under the peak between FWHMWav1 and
FWHMWav2
    realtype    Area;       // The total area under the peak
    realtype    CentroidWav; // The center wavelength of the area of the peak
    realtype    CentroidAmp; // The amplitude at wavelength CentroidWav
    realtype    PolyFitCtrWav; // Center wavelength by a Polynomial fit
    realtype    PolyFitCtrAmp; // Amplitude at the center wavelength by a
Polynomial fit
    realtype    PolyFitA20; // First coefficient (degree 0) of the polymial
fit
    realtype    PolyFitA21; // Second coefficient (degree 1) of the polymial fit
    realtype    PolyFitA22; // Third coefficient (degree 2) of the polymial fit

    int    iPeak; // wavelength index of WaveLength, internal use only
    int    iBase1; // wavelength index of BaseWav1, internal use only
    int    iBase2; // wavelength index of BaseWav2, internal use only
    int    iCentroidWav; // wavelength index of CentroidWav, internal use only
    int    iFWHM1; // wavelength index of FWHMWav1, internal use only
    int    iFWHM2; // wavelength index of FWHMWav2, internal use only
    int    iFWHMCtrWav; // wavelength index of FWHMCtrWav, internal use only
} PEAK_TYPE;
```

The ARRAY_OF_PEAK_TYPE structure is used to build an array of the PEAK_TYPE structures to return multiple peaks using a single pointer. The members are described below.

```
typedef struct
{
    int          NumPeaks; // The number of peaks returned in PeaksDet
    PEAK_TYPE    PeaksDet[MAX_PEAKS+1]; // A PEAK_TYPE structure for each peak
detected. The peaks are arranged in the order of ascending wavelength.
} ARRAY_OF_PEAK_TYPE;
```

CDI OnCalcSavGolay

Syntax: CDI_OnCalcSavGolay (int SavitskyMethod, int convNumPts, BOOL OnlineSavitsky)

Parameters

SavitskyMethod int variable which specifies the algorithm, as described below:

SavitskyMethod = 0:	Smooth, Degree 2 or 3
SavitskyMethod = 1:	Smooth, Degree 4 or 5
SavitskyMethod = 2:	First Derivative, Degree 2
SavitskyMethod = 3:	First Derivative, Degree 3 or 4
SavitskyMethod = 4:	First Derivative, Degree 5 or 6
SavitskyMethod = 5:	Second Derivative, Degree 2 or 3
SavitskyMethod = 6:	Second Derivative, Degree 4 or 5

convNumPts int variable which specifies the number of points used in the algorithm

OnlineSavitsky BOOL variable which when TRUE enables the automatic calculation of the algorithm for each subsequent spectrum acquired by the DLL.

Return Value No return value.

Comments: Windows NT / 2000 ONLY. Not available in Windows 9.x.

CDI_GetBufferedSpectraAbsolute

Syntax: `CDI_GetBufferedSpectraAbsolute (FILETIME *RefTimeStamp,
int maxFileTimes,
FILETIME *FileTimeBuffer,
int maxFloats,
float *SpecDataBuffer,
int *FloatsPerSpectrum,
int *initBufferIndex)`

Parameters

RefTimeStamp	Input pointer to a FILETIME structure which specifies the reference time. All spectra will be returned which have a timestamp later than the reference time.
MaxFileTimes	Input int variable specifying the size of the return buffer FileTimeBuffer in units of FILETIME structures.
FileTimeBuffer	Pointer to a FILETIME structure used to return the timestamps for all the spectra. The timestamps are stored in ascending order.
MaxFloats	Input int variable specifying the size of the return buffer SpecDataBuffer in units of single precision floating point values.
SpecDataBuffer	Pointer to the buffer in which the spectra are returned in single precision floating point format. The spectra are stored in the order of ascending timestamps.
FloatsPerSpectrum	Pointer to the int variable in which is returned the number of single precision floating point values in each spectrum.
InitBufferIndex	Pointer to the int variable in which is returned the index of the first spectrum returned (for diagnostic purposes).
Return Value	The number of spectra returned in SpecDataBuffer.
Comments:	Windows NT / 2000 ONLY. Not available in Windows 9.x.