Definitions for vss_bitflow.dll

All functions are declared as int; the returned integer is always an error code. 0 indicates no error. Error codes are as follows:

Errors:

Error	Description
5000	Error finding the specified board type and number installed in the system.
5001	Error opening board.
5002	The board must be opened before calls to other BitFlow VI’s are made.
5003	Unable to assign memory to Buffers.
5004	Error Setting up Acquisition.
5005	Acquisition setup must be called first before calling this vi.
5006	Error waiting for frame. A possible cause could be a timeout, try increasing the timeout value in the camera file.
5007	The grab command failed.
5008	The board did not close properly.
5009	Error aborting acquisition.
5010	Error resetting acquisition.
5011	There must be at least two buffers allocated for acquisition.
5012	Error while performing Buffer Cleanup.
5013	Buffer Setup must be called first before calling this vi.
5014	Invalid Aqcuisition Engine.
5015	Error setting Frame Size.
5016	Register not on this board.
5017	Error in Register Poke.
5018	Error getting Register Name.
5019	Error Reading from Register.
5020	Error Writing to Register.
5021	Error getting buffer status.
5022	Function not supported for sequential acquisition mode.
5023	Error Reading NTG values.
5024	Error getting the trigger mode.
5025	Error setting the trigger mode.
5026	Error weaving images into the MultiBuffer.
6000	Error assigning memory to Buffers.
6001	Error Setting up Acquisition.
6002	Error Stopping Acquisition.
6003	Invalid Acquisition mode choice.
6004	Acquisition mode needs to be set to Sequential.
7000	Error initializing CL Serial Port.
7001	Error Getting Port Info.
7002	Error Getting Supported Baud Rates.
7003	Error setting Serial Baud Rate.
7004	Error reading from serial port.
7005	Error Writing to serial port.
7006	Error getting Number of Bytes available.
7007	Error getting number of ports.
7008	Error flushing the port.
7009	Error getting error text.

8001	Error opening Device Handle.
8002	Device Handle not opened.
8003	Error getting number of nodes.
8004	Error getting BFGTL node name.
8005	Error opening BFGTL node.
8006	Error retrieving BFGTL node property.
8007	Error reading BFGTL node value.
8008	Error writing BFGTL node value.
8009	Error closing device handle.

Functions

The updated driver for BitFlow SDK 6.4, or greater, supports opening more than one board, and capturing images from all open boards. As boards are open they are assigned a reference number. In the case of multiple boards, this reference number must be passed into BitFlow VI’s. BrdRef is this input reference number. The reference number is then outputted from the VI. The only exceptions to this are BF_Open and BF_Close. BF_Open will open the appropriate board selected using the Board Open Dialog, then assign a board reference number. There is no reference number input for BF_Open. BF_Close only requires the input reference number.

```
int DLLEXPORT BF_Open(int *BrdRef, BFBOOL useBrdOpenDialog, int BrdNum, PBFCHAR camFile,
PBFU32 BrdType);
```

Opens a board for access. This function must return successfully before any other functions are called.

- **UseBrdOpenDialog: [IN]** If true, the Board open dialog is used to select the board, otherwise BrdNum is used.
- **BrdNum: [IN]** If useBrdOpenDialog is set to false, then BrdNum Specifies the board number to open.
- **CamFile: [IN]** The camera file to open. The camera file should include the name and the file extension.(Optional)
- **BrdRef: [OUT]** Reference number for the board opened.
- **BrdType: [OUT]** Returns the type of board opened.

```
int DLLEXPORT BF_ImageInfo(int BrdRef, PBFU32 xsize, PBFU32 ysize, PBFU32 bytesperpixel,
PBFU32 bitDepth);
```

Returns information about the images generated by the camera.

- **BrdRef: [IN]** Reference number for the board.
- **Xsize: [OUT]** width of image in pixels.
- **Ysize: [OUT]** camera 0 height of image in lines.
- **Bytesperpixel: [OUT]** Depth of pixel in bytes.
- **bitDepth: [OUT]** depth of pixel in bits.

```
int DLLEXPORT BF_BufferSetup(int BrdRef, void** BufPtrArray, int AqMode, int ErrMode,
BFU32 NumBuffers);
```

Sets up the system for acquisition to a set of buffers. Acquisition can be sequential or circular. The BF_BufferCleanup.vi should be called at the end in order to free the resources.

- **BrdRef: [IN]** Reference number for the board.
- **BufPtrArray: [IN]** A pointer to an array of pointers that points to each buffer that has been allocated by the user.
- **AqMode: [IN]** Acquisition Mode. 0: Circular; 1: Sequential.

- **ErrMode: [IN]** Acquisition Error Mode. 0: Stop acquiring images if the buffers are full and unavailable; 1: Continue acquisition by overwriting the buffers regardless of the status.
- **NumBuffers: [IN]** The number of buffers that have been allocated

```
int DLLEXPORT BF_Grab(int BrdRef);
```

Starts the acquisition in the mode specified during buffer setup. BF_BufferSetup must be run prior to this VI.

- **BrdRef: [IN]** Reference number of the board.

```
int DLLEXPORT BF_Wait(int BrdRef, int Timeout);
```

Does an efficient wait for the sequence to be completely captured.

- **BrdRef: [IN]** Reference number for the board.
- **Timeout: [IN]** Number of milliseconds to wait for the sequence to be acquired before returning with a timeout error.

```
int DLLEXPORT BF_Wait4Frame(int BrdRef, BFU32 Timeout, PBFU32 BufNum, PBFTime TimeStamp);
```

Wait for a frame to be acquired. Function returns once the frame has been acquired.

- **BrdRef: [IN]** Reference number for the board.
- **Timeout: [IN]** Number of milliseconds to wait for the sequence to be acquired before returning with a timeout error.
- **BufNum: [OUT]** The number of the buffer into which the latest frame was acquired.
- **TimeStamp: [OUT]** A high-resolution time stamp of when the image finished acquisition into memory.

```
int DLLEXPORT BF_Stop(int BrdRef);
```

Stops image acquisition. Use this to ensure that the framegrabber isn't overwriting the image currently in memory. After acquisition has been stopped, any snap or grabbing of an image will restart acquisition without the need for calling BF_Buffersetup again. Stop is intended for use when in continuous grab mode. In the case of sequential acquistion, once a complete frame has been captured into each buffer, the acquisition stops.

- **BrdRef: [IN]** Reference number for the board.

```
int DLLEXPORT BF_BufferCleanup(int BrdRef);
```

Frees all resources used by the acquisition process. Makes sure the board is in a stable state.

- **BrdRef: [IN]** Reference number for the board.

```
int DLLEXPORT BF_Close(int BrdRef);
```

Closes the framegrabber. If the board is not open, board closed will return false.

- **BrdRef: [IN]** Reference number for the board.

```
int DLLEXPORT BF_Status(int BrdRef, PBFBOOL isGn2, PBFBOOL Start, PBFBOOL   Stop, PBFBOOL
Abort, PBFBOOL Pause, PBFBOOL Cleanup, PBFU32 Captured, PBFU32 Missed, PBFU32
pMajorVersion, PBFU32 pMinorVersion);
```

Returns information about the board, the acquisition status, and the Version number of the DLL.

- **BrdRef:** [IN] Reference number for the board.
- **isGn2:** [OUT] Returns true if the board is Gen2.
- **Start:** [OUT] Returns the start status of acquisition. If Start is TRUE, the acquisition of image data to buffers has started. If Start is FALSE, acquisition of image data has either been stopped, never started, or aborted.
- **Stop:** [OUT] Returns the stop status of acquisition. If Stop is TRUE, acquisition of image data to buffers has been stopped, aborted, or never started. When acquisition is stopped, the last frame is fully acquired, then acquisition is stopped. If Stop is FALSE, image data is being acquired.
- **Abort:** [OUT] Returns the abort status of acquisition.If Abort is TRUE, acquisition of image data to buffers has been aborted. When acquisition is aborted, acquisition of data is stopped immediately, not waiting for the last frame to be completely acquired. If Abort is FALSE, acquisition has not been aborted.
- **Pause:** [OUT] Returns the pauses status of acquisition. If Pause is TRUE, acquisition of image data to buffers has been paused. If FALSE, acquisition has not been paused.
- **Cleanup:** [OUT] Returns the clean up status. If Cleanup is TRUE, BiSeqCleanUp or BiCircCleanUp has been called. If FALSE, BiSeqCleanUp or BiCircCleanUp has not been called.
- **Captured:** [OUT] Returns the number of frames that have been captured.
- **Missed:** [OUT] Returns the number of frames that have been missed.
- **PmajorVersion:** [OUT] The major version number.If the highest order bit is set, the DLL is a debug version.
- **PminorVersion:** [OUT] The minor version number.

```
int DLLEXPORT BF_RegNametoId(int BrdRef, int *RegId, char *pRegName, BFBOOL flag);
```

Convert between register name and ID, depending on the value of “flag.”

- **BrdRef:** [IN] Reference number for the board.
- **RegId:** [IN/OUT] pointer to Register ID storage.
- **pRegName:** [IN/OUT] pointer to register name storeage.
- **flag:** [IN] If set to 0, then use name to return the RegID, else use the RegID to return the name.

```
int DLLEXPORT BF_RegPeek(int BrdRef, BFU32 RegId, int *RegValue);
```

This provides access for viewing any register on the board. Refer to the hardware manual for information on registers. The BF_RegNametoId() function can be used to convert between register name and ID.

- **BrdRef:** [IN] Reference number of the board.
- **RegID:** [IN] Register ID
- **RegValue:** [OUT] The bitfield value.

`int DLLEXPORT BF_RegPoke(int BrdRef, BFU32 RegId, BFU32 RegValue);`

Sets the value of register RegId to RegValue. Refer to the hardware manual for information on registers. The BF_RegNametoId() function can be used to convert between register name and ID.

- **BrdRef: [IN]** Reference number of the board.
- **RegID: [IN]** Register ID
- **RegValue: [IN]** Value to write in to the register.

`int DLLEXPORT BF_ExposureControlSet(int BrdRef, PBFDOUBLE Exposure_ms, PBFDOUBLE Period_ms, PBFU32 TriggerMode, PBFU32 OutputSignal, PBFBOOL AssertHigh);`

Programs the board's timing generator, used to create waveforms to control the line/frame rate and exposure time of cameras.

- **BrdRef: [IN]** Reference number of the board.
- ExposurePeriod: [IN] The desired exposure period in milliseconds
  Note: This parameter is floating point and you can pass in non-whole number values (e.g. 10.523)
- LineFramePeriod: [IN] The desired line/frame rate period in milliseconds.
  Note: This parameter is floating point and you can pass in non-whole number values (e.g. 10.523)
- TriggerMode: [IN] The triggering mode for the timing generator. Must be one of the following:
  - FreeRun – Timing generator is free running.
  - OneShotTrigger – Timing generator is in one-shot mode, triggered by the board's trigger input.
  - OneShotEncoder – Timing generator is in one-shot mode, triggered by the board's encoder input.
- AssertedHigh: [IN] The level of the timing generator's output waveform. Must be:
  - TRUE – Waveform is asserted high.
  - FALSE – Waveform is asserted low.
- OutputSignal: [IN] The outputs that the waveform will be output on. Can be one or more of the following OR-ed together (signal will be output on all pins selected by this parameter):
  - For the Karbon/Neon/Alta:
    - BFNTGOutputCC1 – Output on the CC1 signal on CL connector.
    - BFNTGOutputCC2 – Output on the CC2 signal on CL connector.
    - BFNTGOutputCC3 – Output on the CC3 signal on CL connector.
    - BFNTGOutputCC4 – Output on the CC4 signal on CL connector.
    - BFNTGOutputGP0 – Output on GPOUT0 on the I/O connector.
    - BFNTGOutputGP1 – Output on GPOUT1 on the I/O connector.
    - BFNTGOutputGP2 – Output on GPOUT2 on the I/O connector.
    - BFNTGOutputGP3 – Output on GPOUT3 on the I/O connector.
    - BFNTGInputTrig – Output goes to Trigger input.
    - BFNTGInputEncA – Output goes to Encoder A input.
  - For the Aon/Axion/Cyton
    - BFNTGOutputCC1 – Output on the CC1 signal.
    - BFNTGOutputCC2 – Output on the CC2 signal.
    - BFNTGOutputCC3 – Output on the CC3 signal.
    - BFNTGOutputCC4 – Output on the CC4 signal.
    - BFNTGInputTrig – Output goes to Trigger input.
    - BFNTGInputEncA – Output goes to Encoder A input.
    - BFNTGInputEncB – Output goes to Encoder B input.

```
int DLLEXPORT BF_ExposureControlGet(int BrdRef, BFDOUBLE Exposure_ms, BFDOUBLE Period_ms,
BFU32 TriggerMode, BFU32 OutputSignal, BFBOOL AssertHigh);
```

Retrieve the current parameters of the timing generator.

- **BrdRef: [IN]** Reference number of the board.
- **ExposurePeriod: [OUT]** returns the current exposure period in milliseconds.
  Note: This parameter is floating point and you can pass in non-whole number values (e.g. 10.523)
- **LineFramePeriod: [OUT]** returns the current line/frame rate period in milliseconds.
  Note: This parameter is floating point and you can pass in non-whole number values (e.g. 10.523)
- **TriggerMode: [OUT]** returns the current triggering mode for the timing generator. Will be one of the following:
  - FreeRun – Timing generator is free running.
  - OneShotTrigger – Timing generator is in one-shot mode, triggered by the board’s trigger input.
  - OneShotEncoder – Timing generator is in one-shot mode, triggered by the board’s encoder input.
- **AssertedHigh: [OUT]** returns the current the current level of the timing generator’s output waveform. Will be:
  - TRUE – Waveform is asserted high.
  - FALSE – Waveform is asserted low.
- **OutputSignal: [OUT]** returns the current outputs that the waveform is being output on. Will be one or more of the following ORed together: (signal will be output on all pins selected by this parameter):
  - For the Karbon/Neon/Alta:
    - BFNTGOutputCC1 – Output on the CC1 signal on CL connector.
    - BFNTGOutputCC2 – Output on the CC2 signal on CL connector.
    - BFNTGOutputCC3 – Output on the CC3 signal on CL connector.
    - BFNTGOutputCC4 – Output on the CC4 signal on CL connector.
    - BFNTGOutputGP0 – Output on GPOUT0 on the I/O connector.
    - BFNTGOutputGP1 – Output on GPOUT1 on the I/O connector.
    - BFNTGOutputGP2 – Output on GPOUT2 on the I/O connector.
    - BFNTGOutputGP3 – Output on GPOUT3 on the I/O connector.
    - BFNTGInputTrig – Output goes to Trigger input.
    - BFNTGInputEncA – Output goes to Encoder A input.
  - For the Aon/Axion/Cyton
    - BFNTGOutputCC1 – Output on the CC1 signal.
    - BFNTGOutputCC2 – Output on the CC2 signal.
    - BFNTGOutputCC3 – Output on the CC3 signal.
    - BFNTGOutputCC4 - Output on the CC4 signal.
    - BFNTGInputTrig - Output goes to Trigger input.
    - BFNTGInputEncA - Output goes to Encoder A input.
    - BFNTGInputEncB - Output goes to Encoder B input.

```
int DLLEXPORT BF_AqControl(int BrdRef, BFU32 Command, BFU32 Options);
```

Controls the acquisition system.

- **BrdRef: [IN]** Reference number for the board.
- **Command: [IN]** Acquisition command to initiate:
  - BISTART - Starts circular acquisition.
  - BISTOP - Stops circular acquisition after the current frame has been acquired.
  - BIPAUSE - Pauses circular acquisition after the current frame has been acquired.
  - BIRESUME - Resumes circular acquisition after a pause command.
  - BIABORT - Stops circular acquisition immediately.Does not wait for the current frame to be acquired.
- **Options: [IN]** control options for sequence capture are:
  - **BiWait** - wait for the current command to complete.
  - **BiAsync** - as soon as the command is issued return..

```
int DLLEXPORT BF_GetStageMode(int BrdRef, PBFU32 multiImgWidth, PBFU32 multiImgHeight);
```

The LED_XXXX bits are used to determine which stage mode the camera is in. These bits are set by the BFML file. The BFML programs the LED bits, and we read the LED bits and convert to stage mode seting. The LED_XXXX bits are combined to form a 4 bit word.

- **BrdRef: [IN]** Reference number of the board.
- **MultiImgWidth: [OUT]** Number of images to be captured in the X-direction.
- **multiImgHeight: [OUT]** Number of images to be captured in the Y-direction.

BitFlow GenTL Functions:

```
int DLLEXPORT BFGTL_getNumNodes(int BrdRef, size_t *nodeCnt);
```

This provides a method for retrieving the number of GenTL interface nodes in the device. Refer to the Camera’s manual to get a list of the available nodes.

- **BrdRef: [IN]** Index number for the board.
- **nodeCnt: [OUT]** Number of nodes found.

```
int DLLEXPORT BFGTL_getNode(int BrdRef, char *nodeName, size_t nodeIdx);
```

This provides a method for retrieve the name of a node by its internal index. Refer to the Camera’s manual to get a list of the available nodes.

- **BrdRef: [IN]** Index number for the board.
- **NodeIdx: [IN]** Index value of the node to retrieve.
- **NodeName: [OUT]** Name of the node at nodeIdx.

```
int DLLEXPORT BFGTL_getNodeProperty(int BrdRef, char *nodeName, BFGTLNodeField nodeField,
unsigned long *nodeFieldVal);
```

Retrieve the value of a node property.

- **BrdRef: [IN]** Index number for the board.
- **NodeName: [IN]** Name of the node at nodeIdx.
- **NodeField: [IN]** The GenTL field to inquire upon.
- **NodeFieldVal: [OUT]** The return pointer of the property value.

Note:

nodeField can be:

```
BFGTL_NODE_TYPE               = 0x0000, The node's BFGTLNodeType.
BFGTL_NODE_ACCESS             = 0x0001, The node's BFGTLAccess.
BFGTL_NODE_VISIBILITY         = 0x0006, The node's BFGTLVisibility.
```

BFGTL_NODE_TYPE:

```
    BFGTL_NODE_TYPE_UNKNOWN     = 0x0000,
    BFGTL_NODE_TYPE_VALUE       = 0x0001,
    BFGTL_NODE_TYPE_BASE        = 0x0002,
    BFGTL_NODE_TYPE_INTEGER     = 0x0003,
    BFGTL_NODE_TYPE_BOOLEAN     = 0x0004,
    BFGTL_NODE_TYPE_COMMAND     = 0x0005,
    BFGTL_NODE_TYPE_FLOAT       = 0x0006,
    BFGTL_NODE_TYPE_STRING      = 0x0007,
    BFGTL_NODE_TYPE_REGISTER    = 0x0008,
    BFGTL_NODE_TYPE_CATEGORY    = 0x0009,
    BFGTL_NODE_TYPE_ENUMERATION = 0x000A,
    BFGTL_NODE_TYPE_ENUM_ENTRY  = 0x000B,
    BFGTL_NODE_TYPE_PORT        = 0x000C
```

BFGTL_NODE_ACCESS:

```
    BFGTL_ACCESS_UNKNOWN    = 0,
    BFGTL_ACCESS_NI         = 1,    // Not Implemented
    BFGTL_ACCESS_NA         = 2,    // Not Accessible
    BFGTL_ACCESS_RO         = 3,    // Read Only
    BFGTL_ACCESS_WO         = 4,    // Write Only
    BFGTL_ACCESS_RW         = 5     // Read/Write
```

```
BFGTL_NODE_VISIBILITY:
   BFGTL_VISIBILITY_UNKNOWN    = 0,
   BFGTL_VISIBILITY_BEGINNER   = 1,
   BFGTL_VISIBILITY_EXPERT     = 2,
   BFGTL_VISIBILITY_GURU       = 3,
   BFGTL_VISIBILITY_INVISIBLE  = 4
```

```
int DLLEXPORT BFGTL_getNodeValue_str(int BrdRef, char* nodeName, char *valStr, size_t
*iSize);
```

Retrieve the value of the device node as a string.

- **BrdRef: [IN]** Index number for the board.
- **NodeName: [IN]** Name of the node at nodeIdx.
- **ValStr: [OUT]** The node field value as a string.
- **Isize: [OUT]** buffer byte size required to read the entire data value.

```
int DLLEXPORT BFGTL_setNodeValue_str(int BrdRef, char* nodeName, char *valStr, size_t
length);
```

Write the value of the device node using a string.

- **BrdRef: [IN]** Index number for the board.
- **NodeName: [IN]** Name of the node at nodeIdx.
- **valStr: [IN]** The node field value as a string.
- **length: [IN]** Buffer byte size required to read the entire data value.

Note: If BFGTL_NODE_TYPE_COMMAND, use BFGTL_executeCommand to set values.

```
int DLLEXPORT BFGTL_executeCommand(int BrdRef, char* nodeName);
```

Sets the value of a BFGTL_NODE_TYPE_COMMAND.

- **BrdRef: [IN]** Index number for the board.
- **NodeName: [IN]** Name of the node at nodeIdx.

```
int DLLEXPORT BFGTL_getEnumChildren(int BrdRef, char * nodeName, char *enumTable, size_t
*iSize);
```

Retrieve the child entries if ENUM type.

- **BrdRef: [IN]** Index number for the board.
- **NodeName: [IN]** Name of the node at nodeIdx.
- **enumTable: [OUT]** Returns all the ENUM children along with their offsets as a string.
- **iSize: [OUT]** Buffer byte size required to read the entire data value.

CameraLink Serial Comm:

```
int DLLEXPORT CL_SerialInit(CLUINT32 serialIndex, hSerRef* serialRefPtr);
```

Initializes the device referred to by serialIndex, and returns a pointer to an internal serial reference structure.

- **serialIndex: [IN]** The number of the serial device in the system to initialize. This number is a zero-based index value. This n number of serial devices in the system, the serialIndex has a range 0 to (n-1).
- ***serialRefPtr: [OUT]** Points to a value that contains, on a successful call, a pointer to the vendor-specific reference to the current session.

```
int DLLEXPORT CL_GetPortInfo(CLUINT32 serialIndex, CLINT8 * manufacturerName,
CLINT8 * portID, CLUINT32 * version);
```

This function provides information about the port specified by *serialIndex*.

- **serialIndex: [IN]** Zero based index of the serial port you are finding the name for. Use CL_GetNumPorts to determine the valid range of this parameter. This range will be 0 to numSerial-
- Ports-1.
- ***manufacuterName: [OUT]** Pointer to a user allocated buffer into which the function copies the manufacturer name. The returned name is NULL terminated.
- ***portID: [OUT]** The identifier for the port.
- ***version: [OUT]** The version of the Camera Link specifications with which the framegrabber complies.

```
int DLLEXPORT CL_GetSupportedBaudRates(hSerRef serialRef, CLUINT32 *BaudRates);
```

This function returns the valid baud rates that the framegrabber supports for serial communication.

- **serialRef: [IN]** The serial reference returned by clSerialInit.
- ***baudRates: [IN]** Indicates which baud rates are supported by the framegrabber. This is represented as a bitfield with the following constants:
  - CL_BAUDRATE_9600 - 9600 baud rate. Value = 1.
  - CL_BAUDRATE_19200 - 19200 baud rate. Value = 2.
  - CL_BAUDRATE_38400 - 38400 baud rate. Value = 4
  - CL_BAUDRATE_57600 - 57600 baud rate. Value = 8.
  - CL_BAUDRATE_115200 - 115200 baud rate. Value = 16
  - CL_BAUDRATE_230400 - 230400 baud rate. Value = 32.
  - CL_BAUDRATE_460800 - 460800 baud rate. Value = 64.
  - CL_BAUDRATE_921600 - 921600 baud rate. Value = 128.

```
int DLLEXPORT CL_SetBaudRate(hSerRef serialRef, CLUINT32 BaudRate);
```

This function sets the baud rate for the serial port on the framegrabber.

- **serialRef: [IN]** The serial reference returned by clSerialInit.
- **BaudRate: [IN]** Indicates which baud rates are supported by the framegrabber. This is represented as a bitfield with the following constants:
  - CL_BAUDRATE_9600 - 9600 baud rate. Value = 1.
  - CL_BAUDRATE_19200 - 19200 baud rate. Value = 2.
  - CL_BAUDRATE_38400 - 38400 baud rate. Value = 4
  - CL_BAUDRATE_57600 - 57600 baud rate. Value = 8.

  - CL_BAUDRATE_115200 - 115200 baud rate. Value = 16
  - CL_BAUDRATE_230400 - 230400 baud rate. Value = 32.
  - CL_BAUDRATE_460800 - 460800 baud rate. Value = 64.
  - **CL_BAUDRATE_921600 - 921600 baud rate. Value = 128.**

```
int DLLEXPORT CL_SerialRead(hSerRef serialRef, CLINT8* buffer, CLUINT32* bufferSize,
CLUINT32 serialTimeout);
```

This function reads the serial device referenced by serialRef.
Note: Deprecated as of CL 2.1.

- ***serialRef:** [IN] The value obtained from the clSerialInit function.
- ***buffer:** [OUT] Points to a user-allocated buffer. Upon a successful call, buffer contains the data read from the serial device. If there is an error or timeout, the buffer will be returned empty.
- ***bufferSize:** [IN] The number of bytes requested by the caller.
- **SerialTimeout:** [IN] Indicates the time-out in milliseconds.

```
int DLLEXPORT CL_SerialReadEx(hSerRef serialRef, CLINT8* buffer, CLUINT32* numBytes,
CLUINT32 serialTimeout);
```

This function reads the serial device referenced by serialRef.

- ***serialRef:** [IN] The value obtained from the clSerialInit function.
- ***buffer:** [OUT] Points to a user-allocated buffer. Upon a successful call, buffer contains the data read from the serial device. If there is an error or timeout, the buffer will be returned empty.
- ***bufferSize:** [IN] The size of the buffer in bytes. Upon a successful call contains the number of bytes read from the device.
- **SerialTimeout:** [IN] Indicates the time-out in milliseconds.

```
int DLLEXPORT CL_SerialWrite(hSerRef serialRef, CLINT8* buffer, CLUINT32 *bufferSize,
CLUINT32 serialTimeout);
```

Writes the data in the buffer to the serial device referenced by serialRef.

- ***serialRef: [IN]** The value obtained from the clSerialInit function.
- ***buffer: [IN]** Contains data to write to the serial port.
- ***bufferSize: [IN]** Contains the buffer size indicating the maximum number of bytes to be written. Upon a successful call, bufferSize contains the number of bytes written to the serial device.
- **serialTimeout**: [IN] Indicates the time-out in milliseconds.

```
int DLLEXPORT CL_GetNumBytesAvailable(hSerRef serialRef, CLUINT32* numBytes);
```

This function outputs the number of bytes that are received, but not yet read out.

- **serialRef: [IN]** The serial reference returned by clSerialInit.
- ***numBytes: [OUT]** The number of bytes currently available to be read from the port.

```
int DLLEXPORT CL_GetNumPorts(CLUINT32* Ports);
```

This function returns the number of Camera Link serial ports installed in the computer that are supported by clallserial.dll.

- ***numPorts: [OUT]** The number of Camera Link serial ports installed in the computer.

```
int DLLEXPORT CL_FlushPort(hSerRef serialRef);
```

This function discards any bytes that are available in the input buffer.

- **serialRef: [IN]** The value obtained by the clSerialInit function that describes the port to be flushed.

```
int DLLEXPORT CL_GetErrorText(const CLINT8* manuName, CLINT32 errorCode, CLINT8*
errorText, CLUINT32* errorTextSize);
```

This function converts an error code to error text which can be displayed in a dialog box or in the standard I/O window.

- ***manuName: [IN]** The manufacturer name in a NULL-terminated buffer. Manufacturer name is returned from CL_GetPortInfo.
- **ErrorCode: [IN]** The error code used to look up the appropriate error text. This code can be returned
- from any function in this library.
- ***errorText: [OUT]** A caller allocated buffer which will contain a NULL terminated error description on return.
- ***errorTextSize: [IN/OUT]** As an input, this value is the size, in bytes, of the errorText buffer that is passed in. On success, this value is the number of bytes that have been written into the buffer, including the null termination character. On CL_ERR_BUFFER_TOO_SMALL error, this value is the size of the buffer required to write the data text.

```
int DLLEXPORT CL_SerialClose(hSerRef serialRef);
```

Closes the serial device and cleans up the resources associated with serialRef.

- **SerialRef:[IN]** The value obtained from the clSerialInit function.