StreamStor® Real-Time Storage Controller

StreamStor® Cobra

Software Development Kit (SDK) User's Guide

Copyright and Trademarks

The information in this document is subject to change without notice.

Manual version: 2.9
Publication date: November 2, 2020

License Agreement and Limited Warranty

IMPORTANT: CAREFULLY READ THE TERMS AND CONDITIONS OF THIS AGREEMENT BEFORE USING THE PRODUCT. By installing or otherwise using the StreamStor Product, you agree to be bound by the terms of this Agreement. If you do not agree to the terms of this Agreement, do not install or use the StreamStor Product and return it to Conduant Corporation.

GRANT OF LICENSE. In consideration for your purchase of the StreamStor Product, Conduant Corporation hereby grants you a limited, non-exclusive, revocable license to use the software and firmware which controls the StreamStor Product (hereinafter the "Software") solely as part of and in connection with your use of the StreamStor Product. If you are authorized to resell the StreamStor Product, Conduant Corporation hereby grants you a limited non-exclusive license to transfer the Software only in conjunction with a sale or transfer by you of the StreamStor Product controlled by the Software, provided you retain no copies of the Software and the recipient agrees to be bound by the terms of this Agreement and you comply with the RESALE provision herein.

NO REVERSE ENGINEERING. You may not cause or permit, and must take all appropriate and reasonable steps necessary to prevent, the reverse engineering, decompilation, reverse assembly, modification, reconfiguration or creation of derivative works of the Software, in whole or in part.

OWNERSHIP. The Software is a proprietary product of Conduant Corporation which retains all title, rights and interest in and to the Software, including, but not limited to, all copyrights, trademarks, trade secrets, know-how and other proprietary information included or embodied in the Software. The Software is protected by national copyright laws and international copyright treaties.

TERM. This Agreement is effective from the date of receipt of the StreamStor Product and the Software. This Agreement will terminate automatically at any time, without prior notice to you, if you fail to comply with any of the provisions hereunder. Upon termination of this Agreement for any reason, you must return the StreamStor Product and Software in your possession or control to Conduant Corporation.

LIMITED WARRANTY. This Limited Warranty is void if failure of the StreamStor Product or the Software is due to accident, abuse or misuse.

Hardware: Conduant's terms of warranty on all manufactured products is one year from the date of shipment from our offices. After the warranty period, product support and repairs are available on a fee paid basis. Warranty on all third party materials sold through Conduant, such as chassis, disk drives, PCs, bus extenders, and drive carriers, is passed through with the original manufacturer's warranty. Conduant will provide no charge service for 90 days to replace or handle repair returns on third party materials. Any charges imposed by the original manufacturer will be passed through to the customer. After 90 days, Conduant will handle returns on third party material on a time and materials basis.

Software: The warranty on all software products is 90 days from the date of shipment from Conduant's offices. After 90 days, Conduant will provide product support and upgrades on a fee paid basis. Warranties on all third

party software are passed through with the original manufacturer's warranty. Conduant will provide no charge service for 90 days to replace or handle repair returns on third party software. Any charges imposed by the manufacturer will be passed through to the customer.

DISCLAIMER OF WARRANTIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CONDUANT CORPORATION DISCLAIMS ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT, WITH REGARD TO THE STREAMSTOR PRODUCT AND THE SOFTWARE.

SOLE REMEDIES. If the StreamStor Product or the Software do not meet Conduant Corporation's Limited Warranty and you return the StreamStor Product and the Software to Conduant Corporation, Conduant Corporation's entire liability and your exclusive remedy shall be at Conduant Corporation 's option, either (a) return of the price paid, if any, or (b) repair or replacement of the StreamStor Product or the Software. Any replacement Product or Software will be warranted for the remainder of the original warranty period.

LIMITATION OF LIABILITIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL CONDUANT CORPORATION BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE STREAMSTOR PRODUCT AND THE SOFTWARE. IN ANY CASE, CONDUANT CORPORATION'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE STREAMSTOR PRODUCT AND THE SOFTWARE. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

RESALE. If you are authorized to resell the StreamStor Product, you must distribute the StreamStor Product only in conjunction with and as part of your product that is designed, developed and tested to operate with and add significant functionality to the StreamStor Product; you may not permit further distribution or transfer of the StreamStor Product by your end-user customer; you must agree to indemnify, hold harmless and defend Conduant Corporation from and against any claims or lawsuits, including attorneys' fees, that arise or result from the use or distribution of your product; and you may not use Conduant Corporation's name, logos or trademarks to market your product without the prior written consent of Conduant Corporation.

ENTIRE AGREEMENT; SEVERABILITY. This Agreement constitutes the complete and exclusive agreement between you and Conduant Corporation with respect to the subject matter hereof and supersedes all prior written or oral agreements, understandings or communications. If any provision of this Agreement is deemed invalid under any applicable law, it shall be deemed modified or omitted to the extent necessary to comply with such law and the remainder of this Agreement shall remain in full force and effect.

GOVERNING LAW. This Agreement is governed by the laws of the State of Colorado, without giving effect to the choice of law provisions therein. By accepting this Agreement, you hereby consent to the exclusive jurisdiction of the state and federal courts sitting in the State of Colorado.

Table of Contents

Chapter 1
Introduction

The StreamStor Software Development Kit

StreamStor recorders offer unmatched recording and playback performance for many applications requiring the capture or playback of high speed real-time data. Conduant makes it easy for system designers to use StreamStor by providing the StreamStor Cobra Software Development Kit (SDK). The SDK includes the programming library, drivers and other software required for the development and deployment of StreamStor based recording/playback systems. This manual is applicable for the Cobra PXI Express based recorder products.

The SDK includes an Application Programming Interface (API) library. This library provides the control interface for StreamStor in the form of DLLs (Dynamic Link Libraries) and include file(s) for Windows development environments. Application software can be developed in any environment capable of utilizing these library functions. This includes the various Windows programming languages such as Visual C++ (including C++/CLI), Visual C# and Visual Basic as well as graphical programming environments such as LabVIEW or MATLAB. The software includes interfaces for both Microsoft .NET and "C" based programming languages.

Installing the Software

Your StreamStor system includes the Software Development Kit as a download. Please see the documentation included in the shipment for download instructions or contact Conduant support. The SDK is provided as a Windows installer (msi).

The software installation procedure will install required device drivers, library files, example programs and all other components of the SDK onto your system.

Always review the `readme.html` file included with the SDK for the latest information not included in this manual.

Software Components

The SDK software components include operating system device drivers, support files, programming libraries and utility programs. The installation provides optional packages for installation depending on your requirements.

Cabled PCI Express software

If your system was ordered with the optional support for cabled PCI Express connectivity you must install the Dolphin ICS software included with the SDK. The Dolphin installer is copied to the StreamStor install folder automatically by the supplied setup program if this option is selected during install. Devices that will be managed exclusively over Ethernet will not need this software installed. See separately provided installation instructions for this software.

Windows Uninstall

The StreamStor SDK can be easily uninstalled in Windows by using the operating systems tools to uninstall software. Consult the documentation for your specific operating system version for more information. You can also select “Uninstall StreamStor SDK” in the StreamStor menu.

Windows Library

The software development kit includes a software library for integration of StreamStor into Windows based user applications. The required DLL file is `ssapi3.dll`. The library file `ssapi3.lib` is also included for linking the DLL classes and functions to a "C" language user program or languages that can support a "C" interface. The required include files for “C” linked programs are `cobraapi.h, cobraerrors.h and cobratypes.h.` Only the `cobraapi.h` file needs to be included directly in a user program. Example programs are included in the SDK. All of the include files are installed automatically by the installation software in the “include” sub-directory. The library file for linking user programs is installed in the “lib” directory. Note that only 64 bit libraries are included.

.NET and “C” Development

This SDK includes a .NET version of this API for compatible development languages and environments (Visual C++, Visual C#, Visual Basic). The library targets .NET version 4.5.2 and is compatible with newer .NET versions (4.6, 4.6.1, 4.6.2, 4.7).

All .NET objects are contained in the namespace: “Conduant.StreamStor3”.

The .NET interface includes enumerable collections where appropriate to allow easier access to lists such as the recordings present on the system. All documentation for .NET classes is done with Visual C# notation and types. Visual C++/CLI and Visual Basic types are compatible and utilize similar constructs.

Functions throughout this document may be referred to using either a “C” name or the .NET naming. In general, C functions use a pre-pended “SS3” or "SS3_" to avoid any name conflicts with other software. The .NET member functions utilize a namespace to delineate names so the functions generally are similar to the “C” counterparts but lack the “SS3” leading characters.

Host Interface

The StreamStor Cobra system can utilize either a cabled PCI Express connection to the host system or it can be controlled over an Ethernet connection. All functionality is available with either interface with only small differences in application programming including the function call used to initially open the system connection. The buffer management is also different when operating over a network connection. Performance when reading data back to the host system over PCI Express will be significantly improved compared to Ethernet.

“C” Strings

The “C” based functions used in this API are designed to use either standard “char” based strings or wide strings using “wchar_t”. If the preprocessor definition “UNICODE” is set the wide character versions of string are expected (LPWSTR or LPCWSTR for const strings). Contact Conduant support for more information.

“C” Data Structures / Enumerations

StreamStor “C” API functions use the following structures. Refer to the end of the Function Reference section for details on each structure and its members. Most functions that return a structure include a version parameter that can be used for backward compatibility by always building with the constant defined as the latest version at build time. The library will always return the version requested if possible.

`S_SS3_STATUS` - Status information
`S_SS3_DEVINFO` - Device information parameters
`S_SS3_SWREV` - Various device version strings

Interlaken "ODI" Recorder

The StreamStor "ODI" recorder implements functions and software to allow recording and playback of Odi data streams. The ODI standard was developed by the AXIe consortium to provide a standardized high speed data interface. More information is available on the AXIe website at www.axiestandard.com.

The StreamStor Cobra system utilizes the "ODI1" for input data streams and the "ODI2" port for playback of a recorded ODI data stream. These ports implement the 12 lane optical connection defined in the AXIe ODI standard. The ODI implementation also conforms to the Interlaken standard.

The StreamStor Odi option implements several “C” API functions and structures for use only with this option pre-configured at the factory. The Odi option implements an Interlaken optical data interconnect for high speed transfer of instrument data for recording or playback.

In .NET environments users should instantiate the class `cOdiRecorder` instead of `cRecorder`. The `cOdiRecorder` class uses an internal `cRecorder` class and manages the packet sizes and other complexities of ODI/Interlaken.

For “C” based implementations, users should utilize only the functions with the "`SS3_Odi`" prepend for any functions that require a device handle

(*SSHANDLE*). A few functions that do not require a device handle such as `SS3GetApiVersion` can also be used. Open the recorder using the `SS3_OdiOpen` or `SS3_OdiNetOpen` functions to provide access to these Odi functions.

When used as an Odi record/playback system the software manages a 2 port recording with the port management hidden from the user. The implementation uses virtual ports implemented in the hardware to capture packet size information from the Interlaken data stream. This allows the recorder to capture packet boundaries during recording and to accurately reproduce the packets during playback.

Chapter 2
Initialization

Finding StreamStor Recorders

StreamStor recorders can be controlled through a cabled PCI Express connection or over Ethernet. If multiple StreamStor recorders are connected on PCI Express the `FindDevices` functions will report the number of devices discovered. The `FindNetDevices` functions will search the local network and return a list of devices found.

Opening StreamStor

Before a StreamStor recorder can be used it must be opened. The .NET languages will attempt to open a device when the object is constructed depending on the type of constructor used.

When using the "C" interface there are two open functions for PCIe connections (`SS3Open`, `SS3_OdiOpen`) and for network connections (`SS3NetOpen` and `SS3_OdiNetOpen`). One of these functions must be called and completed without error to establish a valid handle to a device before calling other control functions.

Once opened, the status of the recorder can be checked by calling one of the status functions (e.g. `IsReady`(), `SS3GetStatus`()) to make sure the device is ready and did not experience a self test or other initialization problem.

Multi-Recorder Operation

Multiple StreamStor devices can be opened simultaneously by the same user program. Functions are provided to select an individual recorder or the recorders may be grouped or daisy chained.

Chapter 3
Operation

Basic Operation

The basic operation of a StreamStor recorder requires only simple controls that operate somewhat like a tape recorder. When recording a single input port and using only a single recording (file) the user needs to only perform a few basic steps. The first step after opening the recorder is to create a new recording using the `SS3CreateRecording` "C" function or the `CreateRecording` member function of the `cRecorder` class. As part of creating the recording you must provide the size of the recording file(s) and the port(s) to be used for recording. The port names generally correspond to the names on the faceplate label such as “ODI1” for the high speed 12 lane optical input. If an existing recording is to be reused and it already has data, it must first be erased with an EraseRecording command. To start recording data call the `SS3Record` (“C”) or `Record` member function. To stop recording simply call the `SS3Stop` (`Stop`) function.

Once a recording has been performed, its length can be checked with the `GetLength` function. To retrieve information from a recording it must first be opened using an `OpenRecording` function call to open the recording for reading. Once this is done the data can be read into a user buffer using the read function. Once done reading, the open recording should be closed before performing other operations.

Data Recording

As described above, a simple recording is started by simply calling the `Record` function. The recording can only be stopped by calling the `Stop` function. Data flow may stop flowing from the data source but the stop function must be called to take the recorder out of record mode. It is also possible that the recorder will exhaust all available storage space in the recording file(s) and it will stop recording in this scenario but the stop function must still be called.

☞ **NOTE:** ***A data acquisition system can stop recording by simply ceasing any writes to the StreamStor data address range. The `SS3Stop` function should still be used to flush all data to the disk drives and to prepare for reading of the data.***

Data Playback / Read

The StreamStor recorder is designed to record data at high speed over its external ports or PCI Express connections. Once recorded, the data can be used in a playback operation back out these same high speed ports or it can be read to memory buffers in the users PC for analysis or other purposes.

In order to read data from the StreamStor system, a recording must first be opened for reading. The `OpenRecording` function takes a recording name and port name as input. For simple cases with only a single recording and single port, the recording name and port name can be NULL.

The `Read` function is used to copy data from the recorder to a buffer. The user can provide an allocated buffer to receive the data. When connected over PCI Express the application may use a buffer provided by the API using the `GetDMABufferAddr` function. This function can be called multiple times for double buffering purposes but there is a limited amount of DMA memory available. Note that this memory allocation is only released when the device is closed so the allocated DMA memory address(es) should be saved for reuse in the application. The advantage of using DMA memory is higher performance since a memory copy is avoided. The read function requires a pointer to a buffer (user allocated or DMA buffer address) along with a size to define the number of bytes to copy. The user must ensure that the memory buffer is at least this large to avoid a buffer overrun. The `Read` function also requires a 64 bit offset into the recorded data. This is the starting point where the data will be read before copying to the users buffer.

The StreamStor system is designed to provide maximum performance when operating on sequential streams of data. For maximum performance it is recommended that reads be performed sequentially as much as possible.

Error Handling

When errors occur when using the .NET interface, an exception is thrown

that includes a descriptive message. These exceptions can be trapped by the application to attempt recovery.

All "C" functions return a simple pass (`SS3_SUCCESS`) or fail (`SS3_FAIL`) return value that can be tested by application software. For more detailed information about the failure and error code can be retrieved by calling the `SS3GetLastError` function. An error string corresponding to the error code can also be retrieved by using the `SS3GetErrorMessage` function.

Chapter 4
External Ports

External Port Overview

The external fiber optic ports on the StreamStor Cobra system can be customized to customer specific requirements. This might include various protocols such as ODI, Interlaken, Aurora or SerialFPDP. In addition the ports can be configured for multiple independent channels using combinations of single or multiple fibers to meet data rate requirements on each channel. The data channels provide several advantages over more traditional methods of capturing data including:

- full isolation of data path from operating system and computer hardware facilitating predictable and repeatable behavior;
- better or additional control over electrical timing and other parameters;
- higher utilization efficiency due to non-shared, low overhead protocols;
- access to hardware interface signals without risk of crashing host computer;
- ability to implement multiple ports to route data from many disparate sources;
- higher data rates than internal buses support;

Port Description and Selection

A list of ports available on a StreamStor system can be listed by first calling `SS3GetPortCount` than calling `SS3GetPortName` to get each port name. The cRecorder class also has an enumerable list of ports ("Ports"). The hardware manual for your StreamStor system should also include a list of ports and the port name generally matches the name(s) imprinted on the faceplate of the board. In order to record from a port you must include it when first creating a new recording so that it will be used when the record operation is started.

Port Description

The hardware manual for your StreamStor controller describes the available ports on your board type.

Chapter 5
Recording Management

Recordings Management Overview

The StreamStor system can store multiple recordings and provides tools to manage these recordings. Although not primarily designed as general purpose storage, the system can be used to capture, store and playback recorded data sets.

To capture new data in a record operation you can reuse an existing recording by first erasing the recording or you can create a new recording if there is sufficient free space available to hold the expected data. Recordings are managed by names, which must be unique from each other and should be less than 256 characters.

Creating a Recording

A new recording is created by calling the `SS3CreateRecording` function. A name must be supplied which must be unique from any other existing recording. The recording name is used as the identifier whenever a read, write or other operation is performed on the recording. Optionally a recording size using all available space can be requested.

When creating a recording you must also specify the port(s) that will be used when recording. If creating a multi-port recording there is a version of the recording creation functions designed for this purpose (see `SS3CreateMultiPortRecording`). The recording system does not have information on the expected data rate for each port being recorded so it is up to the user to create files for each port at an appropriate size.

Opening a Recording

In order to read or write data in a recording you must first open the recording. The `OpenRecording` function is provided for this purpose. It is only possible to open a single recording at any one time so a recording must be closed using `CloseRecording` before opening another recording. There are 2 parameters that identify the recording to be opened, the recording name and the port that was used to record the data. The port name is only required in multi-port recordings. You must also specify

whether the recording is to be opened for read or for write.

Getting Recording Information

Information about the recordings present on the system is provided by a few different functions. The .NET implementation provides an enumerable list of objects ("`Recordings`"). Each object has a "`Name`" string member that is the name of the recording and each object has an associated list of recorded ports ("`Recordings.Ports`") for that recording. For "C" applications the list of recordings can be retrieved using the function `SS3GetRecordingName` with a supplied index (0 based). The number and names of the ports recorded in each individual recording can be queried using the `SS3GetRecordedPortCount` and `SS3GetRecordedPortName` functions. The number of recordings is provided by the `SS3GetRecordingCount` function. The length of a recording can be retrieved by the `SS3GetLength` function.

Example (C#) to iterate through the list of Recordings and recorded ports:

```
// "test" is open cRecorder object
Console.WriteLine("RECORDING LIST");
Console.WriteLine(" Count: {0}", test.Recordings.Count);
foreach (var c in test.Recordings)
{
   Console.WriteLine(" Recording: {0}", c.Name);
   Console.WriteLine(" Ports recorded: {0}", c.Ports.Count);
   foreach (var p in c.Ports)
      Console.WriteLine(" Port: {0}", p);
}
```

Deleting a Recording

Recordings can be deleted from the recorder using the `DeleteRecording` functions. Deleting a recording will delete all data from all recorded ports. Note that a recording can be erased and reused instead of deleting. Recordings can also be renamed at any time.

Chapter 6
ODI Recording

Overview

When using the ODI recorder there is both a .NET class (`cOdiRecorder`) and a separate set of "C" functions specifically implemented for this purpose. This implementation manages the multi-port recording operation and provides a simplified interface for managing the recordings and the packets associated with these recordings.

When using the ODI class or "C" functions, other functions should not be used.

Writing / Reading packets

One of the primary purposes of the ODI recorder is to facilitate the management of packets that are used for input and out on the Interlaken optical interface. In order to manage these packets the system has internal software and hardware to capture and reproduce the packet boundaries. For this reason new functions have been created to read or write these packets so that the software can capture and reproduce these boundaries when moving them to/from the user application memory buffers.

The `ReadPacket` and `WritePacket` functions have been developed to provide this capability and allow the system to capture and reproduce the packets. The system will buffer larger collections of packets internally by assuming that reading and writing of packets will happen sequentially. Non- sequential accesses are supported but will incur larger performance penalties. When reading packets the user should supply a buffer large enough to hold the largest possible packet. The function will return the actual size of the packet written to the users buffer space. When writing packets, the user must specify the size of the packet being written. If multiple packets are being read or written, the packet size must be fixed for all packets in a single function call.

Recording Management

The management of recordings is no different from a standard StreamStor recording with the exception that the multi-port nature of an ODI recording is hidden from the user to simplify the interface. The addition of a few functions specific to ODI recordings such as the `GetPacketCount` function provide the required access to the underlying implementation.

External Ports

The Odi recording systems always use the ODI1 port on the front panel for incoming data (record) and ODI2 for all outgoing data (playback). This allows both a digitizer to be connected to the recorders on ODI1 while still allowing the connection of an Arbitrary Waveform Generator to be connected to ODI2. This makes it possible to record and playback analog signals without changing any optical connections.

The external ports have a number of commands that can be accessed as defined on the ODI software specification. The ports are accessed in a list from the cOdiRecorder class (Odi.Ports). The list key is the port name string ("ODI1" or "ODI2"). For example, to activate the ODI1 port you would call the activate this way:

```
recorder.Odi.Ports["ODI1"].Activate(LaneRate.RATE_14R1G,2048,FlowCont
rol.INBAND, null)".
```

Other functions for managing the ports are available using the same technique. See the examples for more information.

The Conduant ODI recording system utilizes 4 Cobra recorders working in parallel to record the 20 GB/s data rate possible with ODI. The ODI1 port on the front panel of the left-most Cobra board in the chassis is the input port for ODI recording and the ODI2 port on the right most system in the chassis is the output port for ODI playback.

Note that not all ODI options are supported on these ports. Currently supported features include:

- Lane Rate: 14.1 Gbps
- Max burst size: 2048 bytes
- Flow control: INBAND

Contact Conduant support for the latest information on supported ODI parameters.

Chapter 7
"C" API Reference

Functions / Structures

“C” Basic Function Reference

These functions are intended for use in environments that cannot make use of the .NET classes and are not using a specialized instruction set such as the ODI functionality. Nearly all functions require a handle (`SSHANDLE`) that is initialized from one of the device open functions (`SS3Open`, `SS3_OdiOpen`, `SS3NetOpen`, `SS3_OdiNetOpen`).

Nearly all functions return `SS3_RETURN_CODE` type which is a success / fail indicator (`SS3_SUCCESS`, `SS3_FAIL`). An error code is saved when a function fails that can be retrieved using `SS3GetLastError` and translated into a description string using `SS3GetErrorMessage`.

SS3ActivatePort

Syntax:

`SS3_RETURN_CODE SS3ActivatePort( SSHANDLE` *`device`*`, LPCTSTR` *`portname`*
`)`

Description:

This function is used to activate or enable an external port. In the case of an optical port, this function will turn on and enable all hardware required for the port to receive and/or playback data.

Parameters:

`device` - device handle returned from a previous call to one of the open functions.

`portname` - port that is to be activated.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3BindOutputPort

Syntax:

```
SS3_RETURN_CODE SS3BindOutputPort( SSHANDLE device, LPCTSTR
recordingname, LPCTSTR recordedport, LPCTSTR newport )
```

Description:

This function modifies the port that is used for playback of a data set. The port used during a record operation is also the port used for data output (Playback) unless changed with this function. This function changes the binding for output to the specified port and the change is persistent across sessions since it is stored with the recorded data. Note that this binds a port for output FROM StreamStor. In other words, "output" is relative to StreamStor. To playback from a particular port that is different from the port used to record the data, the recorded port must be bound to the new output port via this command. This change is captured and stored and will be used in subsequent playback operations. The new name is required in all future references to this recorded data set. Note that if a recording is erased, the output port will revert to the original port used to create the recording.

Parameters:

device - device handle returned from a previous call to one of the open functions.

recordingname - recording to apply the change against.

recordedport - current port name that is to be changed.

newport - new port name that is to be used for playback.

Return Value:

On success, this function returns SS3_SUCCESS. On failure, this function returns SS3_FAIL.

Usage:

See Also:

SS3ClearWriteProtect

Syntax:

```
SS3_RETURN_CODE SS3ClearWriteProtect( SSHANDLE device, LPCTSTR recordingname )
```

Description:

`SS3ClearWriteProtect` removes write protection from a previously write protected StreamStor recording. By default, recordings are not write protected. The system must be idle (i.e., not in record mode or playback mode) to clear the write protection.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`recordingname` - name of the recording to clear write protect.

Return Value: On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

```
SSHANDLE device; SS3_RETURN_CODE status;

// Open the device. status = SS3Open( 1, &device );
   ... status = SS3ClearWriteProtect( device, TEXT("MyRecording1") );
   ... // Close device before exiting. SS3Close( device );
```

See Also:

`SS3SetWriteProtect`

SS3Close

Syntax:

```
void SS3Close( SSHANDLE device )
```

Description:

`SS3Close` closes the StreamStor device. This should be called before exiting an application that has opened a StreamStor device with `SS3Open` or another open function. No other application can open the StreamStor device until this function has been called.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

Return Value:

None.

Usage:

See Also:

`SS3Open`

SS3CloseRecording

Syntax:

`SS3_RETURN_CODE SS3CloseRecording( SSHANDLE device )`

Description:

`SS3CloseRecording` closes an open recording. A recording must be closed when done reading or writing data and before most other types of operations can be performed including playback or record.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3OpenRecording`

SS3CreateMultiPortRecording

Syntax:

SS3_RETURN_CODE SS3CreateMultiPortRecording(SSHANDLE *device*, LPCTSTR *recordingname*, S_PORT_FILE *ports[]*, int *numPorts*, int *structversion*)

Description:

SS3CreateMultiPortRecording will create a new, empty recording on the system and assign the name provided. This function is intended for use when more than one port will be recorded at a time. The provided array of structures provides a list of the ports to be recorded and the size of file to create for each port. The file sizes should be carefully selected based on the data rate of each channel.

Parameters:

device - device handle returned from a previous call to SS3Open.

recordingname – name to be assigned to the new recording.

ports[] - An array of ports and file sizes to use in creating the recording.

size – size of the *S_PORT_FILE* array provided.

structversion – normally should be S_PORT_FILE_VERSION.

Return Value:

On success, this function returns SS3_SUCCESS. On failure, this function returns SS3_FAIL

Usage:

See Also:

SS3CreateRecording

Syntax:

`SS3_RETURN_CODE SS3CreateRecording(SSHANDLE device, LPCTSTR recordingname, LPCTSTR portname, uint64_t size)`

Description:

`SS3CreateRecording` will create a new, empty recording on the system and assign the name provided. The recording file created using this function can only be used with the single port specified. See `SS3CreateMultiPortRecording` if a multiport (channel) recording is required. If the size specified is 0, the system will create the largest possible recording with available unused space.

Parameters:

device - device handle returned from a previous call to `SS3Open`.

name – name to be assigned to the new recording.

portname – name of port to be used when recording data.

size – size (bytes) of recording to create.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`

Usage:

See Also:

SS3DeactivatePort

Syntax:

`SS3_RETURN_CODE SS3DeactivatePort(SSHANDLE` *`device`*`, LPCTSTR` *`portname`*`)`

Description:

This function is used to deactivate or disable an external port. In the case of an optical port, this function will turn off the hardware required for the port to receive and/or playback data.

Parameters:

`device` - device handle returned from a previous call to one of the open functions.

`portname` - port that is to be deactivated.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3DeleteRecording

Syntax:

`SS3_RETURN_CODE SS3DeleteRecording(SSHANDLE` *`device`*`, LPCTSTR recordingname`*`name`*`)`

Description:

`SS3DeleteRecording` will delete the specified recording.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`recordingname` - name of the recording to delete.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3DownloadRecording

Syntax:

`SS3_RETURN_CODE SS3DownloadRecording( SSHANDLE device, LPCTSTR recordingname, LPCTSTR portname, LPCTSTR filepath )`

Description:

`SS3DownloadRecording` can be used to download recorded data directly to a file on the local system.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`recordingname` - name of the recording to download.

`portname` - name of a recorded port in the recording to download.

`filepath` - file to write with downloaded data.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3EraseRecording

Syntax:

`SS3_RETURN_CODE SS3EraseRecording( SSHANDLE device, LPCTSTR recordingname, uint32_t flags )`

Description:

`SS3EraseRecording` erases any data in the specified recording.

This command will erase only the data within the specified recording. Other recordings will be unaffected. Note that all ports that have recorded data will be erased. This function will also revert the recording to the original ports defined when created.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`recordingname` - name of the recording to erase.

`flags` - Not implemented (use 0)

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3FindDevices

Syntax:

```
SS3_RETURN_CODE SS3FindDevices( uint32_t* NumRecorders )
```

Description:

`SS3FindDevice` searches the PCI Express fabric and returns the number of StreamStor devices found. If the SISCI driver is not installed this function will return `SS3_FAIL` and the last error will be set to `SS3_ERR_SISCI_MISSING`.

Parameters:

`NumRecorders` - pointer to unsigned 32 bit integer used to return the number of devices found.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

```
int NumRecorders;

if( SS3FindDevices(&NumRecorders) == SS3_SUCCESS ) {
   if(NumRecorders > 0) {
   // There are StreamStor recorder(s) attached to this system.
   printf("StreamStor recorders found: %d\n", NumRecorders );
} else {
   // No recorders found or no driver present on the system.
   printf("No StreamStor recorders detected!\n");
}
```

See Also:

SS3FindNetDevices

Syntax:

```
SS3_RETURN_CODE SS3FindNetDevices( S_SS3_NETDEVICE_INFO *array,
int arraydim, int structversion, int* count )
```

Description:

`SS3FindNetDevices` searches the local network and returns the number of StreamStor devices found. If a non-null structure array pointer is provided the function will populate the structure(s) with addresses and port numbers of the devices located. The `arraysize` parameter indicates the array size of `S_SS3_NETDEVICE_INFO` structures provided (not number of bytes).

This function can be called with a NULL structure reference to determine the number of structures required. The `NumRecorders` parameter will indicate the size of the `S_SS3_NETDEVICE` array required.

Parameters:

`array` – pointer to array of `S_SS3_NETDEVICE_INFO` structures or `NULL`

`arraydim` – number of structures in above array (ignored if array is NULL).

`structversion` – normally should be `SS3_NETDEVICE_INFO_VERSION`

`count` – pointer to integer to return number of devices discovered.

Return Value:
On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:
`SS3Open`

SS3GetApiVersion

Syntax:

`SS3_RETURN_CODE SS3GetApiVersion( LPTSTR` *`versionstring`*`, int` *`size`* `)`

Description:

`SS3GetApiVersion` returns the API version as a string formatted as a *major.minor* version number.

Parameters:

`versionstring` – pointer to a character string to hold the returned version. It should be at least a length of `SS3_VERSION_LENGTH`.

`size` – size of *`versionstring`* in number of characters.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3GetBaseAddr

Syntax:

`SS3_RETURN_CODE SS3GetBaseAddr( SSHANDLE` *`device`*`, uint64_t*` *`addr`* `)`

Description:

`SS3GetBaseAddr` returns the physical address of the recording data window. This address can be used to program PCI Express hardware devices for direct card-to-card data transfer. The address returned from this function is NOT a valid user address.

NOTE: The capability to perform peer-to-peer data transfers between PCI Express devices requires a configured PCI Express bridge.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`addr` - pointer to a 64 bit integer to hold the requested address

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetBaseRange and SS3GetWindowAddr`

SS3GetBaseRange

Syntax:

`SS3_RETURN_CODE SS3GetBaseRange(SSHANDLE` *`device`*`, uint32_t*` *`range`*`)`

Description:

`SS3GetBaseRange` returns the size of the StreamStor device data window in bytes. This address range is intended to be used by hardware performing peer-to-peer data transfers which cannot be programmed with a non-incrementing address. Note that the address used to write to StreamStor does not affect the storage location of the data; StreamStor always stores data sequentially in the order it is written regardless of the address.

If recording multi-port PCI Express, the total range available is divided between the ports.

NOTE: The capability to perform peer-to-peer data transfers between PCI Express devices requires a configured PCI Express bridge. Contact Conduant support for more information on availability of this capability.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`. *`range`* - memory window size in bytes.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetBaseAddr`, `SS3GetWindowAddr`

SS3GetDeviceInfo

Syntax:

```
SS3_RETURN_CODE SS3GetDeviceInfo( SSHANDLE device, S_SS3_DEVINFO
*devInfo, int structversion )
```

Description:

`SS3GetDeviceInfo` retrieves information from the StreamStor device about its fixed configuration. The function will use the version of the structure supplied in the *`structversion`* parameter to allow backward compatibility if the structure changes.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`pDevInfo` - pointer to an `S_SS3_DEVINFO` structure.

`structversion` - version number of the structure to return. Current version is `SS3_DEVINFO_VERSION`.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3GetDMABufferAddr

Syntax:

`SS3_RETURN_CODE SS3GetDMABufferAddr(SSHANDLE device, void** address, uint32_t sizeBytes)`

Description:

`SS3GetDMABufferAddr` retrieves the address of a DMA buffer provided for reading and writing data to/from a recording. The function will return the address of the buffer as a pointer that can be used in the application. The size of the buffer is specified by the `sizeBytes` parameter. Using this DMA buffer address when provided to the `SS3Write` or `SS3Read` functions will avoid memory copy operations and speed up the transfer. Note that the total memory space available for DMA buffers is limited (usually 64MB).

Parameters:

`device` – device handle returned from a previous call to `SS3Open`.

`address` - address of the DMA buffer.

`sizeBytes` - size of the requested DMA buffer.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3GetDriveCount

Syntax:

`SS3_RETURN_CODE SS3GetDriveCount( SSHANDLE device, int *count )`

Description:

`SS3GetDriveCount` retrieves the number of drives attached to this recorder.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`count` - pointer to int parameter populated with number of disk drives.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetDriveInfo`

SS3GetDriveInfo

Syntax:

```
SS3_RETURN_CODE SS3GetDriveInfo( SSHANDLE device, int drive,
S_SS3_DRIVEINFO* pDriveInfo, int structversion )
```

Description:

`SS3GetDriveInfo` retrieves info from the StreamStor recorder about a specific drive.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`drive` - index number of the drive (must be less the count from SS3GetDriveCount).

`pDriveInfo` - pointer to an `S_SS3_DRIVEINFO` structure.

`structversion` - version of structure to return. Current version is `SS3_DRIVEINFO_VERSION`.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetDriveCount`

SS3GetErrorMessage

Syntax:

```
SS3_RETURN_CODE SS3GetErrorMessage(LPTSTR string, int strsize,
SS3_ERROR_CODE err)
```

Description:

`SS3GetErrorMessage` returns the error message associated with the specified error code. The string will be wide characters if UNICODE is set in the application build settings.

Parameters:

`string` - pointer to a string to accept the error message.

`strsize` - size of the string supplied in number of characters.

`err` – error code to be converted to string message (see `SS3GetLastError`).

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetLastError`

SS3GetFreeCapacity

Syntax:

`SS3_RETURN_CODE SS3GetFreeCapacity(SSHANDLE *device*, uint64_t *\*free*)`

Description:

`SS3GetFreeCapacity` retrieves the amount of space available on the recorder for additional recordings. Note that there is overhead for every recording that will not allow the entire free space to be utilized.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`free` - pointer to parameter to hold remaining capacity available on the recorder (bytes).

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetDeviceInfo`

SS3GetLastError

Syntax:

```
SS3_ERROR_CODE SS3GetLastError( void )
```

Description:

`SS3GetLastError` returns the error code of the most recent API failure. This function should be called immediately after any StreamStor API function call that returns failure.

It is not meaningful to call `SS3GetLastError` if the last StreamStor API function call was successful. In this case, the returned error code may be error code 3 (`SS3_ERR_NOINFO`).

Parameters:

None.

Return Value:

This function returns the error code. Error codes are listed in Appendix A.

Usage:

See Also:

`SS3GetErrorMessage`

SS3GetLength

Syntax:

```
SS3_RETURN_CODE SS3GetLength( SSHANDLE device, LPCTSTR
recordingname, LPCTSTR portname, uint64_t* length)
```

Description:

`SS3GetLength` returns the length (in bytes) of a recording in a 64-bit integer. This function cannot be used during an active recording operation. Note that during active record operations, the function SS3GetProgress should be utilized.

If data on the StreamStor is multi-port, this command will return the length of data recorded on the specified port. If only a single port recording, the portname parameter can be NULL.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`recordingname` - name of the recording

`portname` – name of a recorded port or NULL

`length` - pointer to parameter to hold length of recording in bytes.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetProgress`

SS3GetPlayLength

Syntax:

`SS3_RETURN_CODE SS3GetPlayLength( SSHANDLE` *`device`*`, LPCTSTR` *`recordingname`*`, LPCTSTR` *`portname`*`, uint64_t*` *`length`*`)`

Description:

`SS3GetPlayLength` returns the length (in bytes) of the most recent playback of a recording in a 64-bit integer. The play length is volatile and is only available within the same session as a playback occurred.

If data on the StreamStor is not multi-port, the portname parameter can be NULL.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`recordingname` - name of the recording

`portname` – name of a recorded port or NULL

`length` - pointer to parameter to hold playback length.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3GetPortCount

Syntax:

`SS3_RETURN_CODE SS3GetPortCount(SSHANDLE *device*, int **count*)`

Description:

`SS3GetPortCount` retrieves the number of ports that are available on the system for recording or playback.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`portsRecorded` - pointer to integer to hold the returned number of ports available.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetPortName`

SS3GetPortName

Syntax:

`SS3_RETURN_CODE SS3GetPortName(SSHANDLE` *`device`*`, LPTSTR` *`portname`*`, int` *`strsize`*`, int` *`portnum`*` )`

Description:

`SS3GetPortName` retrieves the name of a port available on the system for recording or playback. The supplied string should be at least `SS3_MAX_NAME` characters in size. This function can be used in combination with `SS3GetPortCount` to retrieve the list of available ports on the system.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`portname` - string to hold the name of the available port.

`strsize` – size of the above string in number of characters.

`portnum` – index number of a port to retrieve name. Must be less than the number of ports returned by `SS3GetPortCount`.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3PortCount`

SS3GetPortStatus

Syntax:

```
SS3_RETURN_CODE SS3GetPortStatus(SSHANDLE device, LPCTSTR portname,
uint32_t *portStatus )
```

Description:

`SS3GetPortStatus` retrieves a status value from the recorder for the named port. The status is a bit significant value. The following are the integer value meanings for each bit in the value returned.

`SS3_M_PORT_ACTIVE` - Indicates the port is powered and activated
`SS3_M_PORT_TX_READY` - Indicates the port is ready to transmit data
`SS3_M_PORT_RX_READY` - Indicates the port is ready to receive data
`SS3_M_PORT_RX_LANE_ERROR` - Indicates the port has experienced an error
`SS3_M_PORT_CRC_ERROR` - Indicate the port had a CRC error in data received
`SS3_M_PORT_RX_OVERRUN` - Indicates the port received that overran its buffer

The status value returned can be compared against these values to indicate if the indicator is true or false (i.e. *portStatus & SS3_M_PORT_ACTIVE)

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`portname` - string to hold the name of the available port.

`strsize` – size of the above string in number of characters.

`portnum` – index number of a port to retrieve name. Must be less than number of ports returned by `SS3GetPortCount`.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3PortCount`

SS3GetProgress

Syntax:

`SS3_RETURN_CODE SS3GetProgress(SSHANDLE *device*, LPCTSTR *portname*, uint64_t **length*)`

Description:

`SS3GetProgress` retrieves the amount of data transferred so far in an active playback or recording in-process.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`portname` - name of the port to retrieve progress information. Can be NULL if single port.

`length` – pointer to 64 bit integer to hold the progress length.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3GetRecordedPortCount

Syntax:

`SS3_RETURN_CODE SS3GetRecordedPortCount(SSHANDLE` *`device`*`, LPCTSTR` *`recordingname`*`, int` *`*count`* `)`

Description:

`SS3GetRecordedPortCount` retrieves the number of ports that have recorded data in the named recording.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`recordingname` - name of the recording to get the port information from

`count – pointer to integer to hold the number of recorded ports`.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3GetRecordedPortName

Syntax:

`SS3_RETURN_CODE SS3GetRecordedPortName(SSHANDLE` *`device`*`, LPCTSTR` *`recordingname`*`, LPTSTR` *`portname`*`, int` *`strsize`*`, int` *`portindex`* `)`

Description:

`SS3GetRecordedPortName` retrieves a portname that has recorded data in the named recording. The `portindex` must fall in the range of recorded ports as returned from `SS2GetRecordedPortCount`. The string returned will be a maximum of `SS3_MAX_NAME` length.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`recordingname` - name of the recording to get the port information from

`portname` – pointer to a string to hold the portname retrieved.

`strsize` – size of the provided portname string in number of characters.

`portindex` – *i*ndex (0 based) of port to get name. Must be less than count returned by `SS3GetRecordedPortCount` function.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetRecordedPortCount`

SS3GetRecorderName

Syntax:

`SS3_RETURN_CODE SS3GetRecorderName(SSHANDLE` *`device`*`, LPTSTR` *`recordername`*`, int` *`*strsize`* `)`

Description:

`SS3GetRecorderName` retrieves the name assigned to the recorder. A default name is assigned at the factory but can be changed (see `SS3SetRecorderName`). If the `recordername` parameter is NULL, the function will update the `strsize` parameter to reflect the string length required to hold the name. If the recordername parameter is not NULL and the size provided is not large enough, the function will return `SS3_FAIL` and set the last error to `SS3_ERR_STRING_SIZE`.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`recordername` - string buffer provided to hold the name.

`strsize` – pointer to integer, if *`recordername`* is NULL returns size of string required, otherwise indicates size of supplied string.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3SetRecorderName`

SS3GetRecorderType

Syntax:

```
SS3_RETURN_CODE SS3GetRecorderType(SSHANDLE device, LPTSTR
recordertype, int *strsize )
```

Description:

`SS3GetRecorderType` This function returns a description of the specific type of StreamStor recorder in use. The type of recorder will vary depending on the type of ports and other customizations implemented.

If the `recordertype` parameter is NULL, the function will update the `strsize` parameter to reflect the string length required to hold the name. If the recordername parameter is not NULL and the size provided is not large enough, the function will return `SS3_FAIL` and set the last error to `SS3_ERR_STRING_SIZE`.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`recordertype` - string buffer provided to hold the type description.

`strsize` – pointer to integer, if *`recordertype`* is NULL returns size of string required, otherwise indicates size of supplied string.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3GetRecordingCount

Syntax:

`SS3_RETURN_CODE SS3GetRecordingCount(SSHANDLE *device*, int **count*)`

Description:

`SS3GetRecordingCount` returns the number of recordings on the recorder.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`count` – pointer to integer to hold returned number of recordings currently stored on the recorder.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3GetRecordingFileSize

Syntax:

`SS3_RETURN_CODE SS3GetRecordingFileSize(SSHANDLE` *`device`*`, LPCTSTR` *`recordingname`*`, LPCTSTR portname, uint64_t` *`*strsize`*`)`

Description:

`SS3GetRecordingFileSize` provides a way to retrieve the size of the recording file created for each port defined in a recording.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`recordingname` - name of the recording.

`portname` - name of the port in the recording.

`filesize` – pointer to varaible to hold returned file size.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3CreateRecording, SS3CreateMultiPortRecording`

SS3GetRecordingName

Syntax:

`SS3_RETURN_CODE SS3GetRecordingName(SSHANDLE` *`device`*`, LPTSTR` *`recordingname`*`, int *`*`strsize`*`, int` *`index`* `)`

Description:

`SS3GetRecordingName` provides a way to retrieve the names of all recordings stored on the recorder. The index can be any value from 0 to the number of Recordings minus one.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`recordingname` - string buffer provided to hold the name.

`strsize` – pointer to integer, if *`recordingname`* is NULL returns size of string in characters required to hold the *`recordingname`*, otherwise indicates size (characters) of supplied string.

`index` – index of the recording name to retrieve.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetRecordingCount`

SS3GetSdkVersion

Syntax:

`SS3_RETURN_CODE SS3GetSdkVersion(LPTSTR versionstring, int strsize)`

Description:

`SS3GetSdkVersion` returns the SDK version as a string formatted as a *major.minor* version number. Note that the SDK is a collection of the various components that make up the software, hardware and firmware of the StreamStor system. The SDK version may not reflect independent updates of these components.

Parameters:

`versionstring` – pointer to a character string to hold the returned version. It must be allocated by the user.

`strsize` – number of characters in the user allocated string, this should be at least `SS3_VERSION_LENGTH`.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetVersion and SS3GetApiVersion`

SS3GetStatus

Syntax:

```
SS3_RETURN_CODE SS3GetStatus( SSHANDLE device, S_SS3_STATUS
*devStatus, int structversion )
```

Description:

`SS3GetStatus` retrieves current status of the StreamStor device.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`devStatus` - pointer to an `S_SS3_STATUS` structure.

`structversion` - version of the structure to retrieve for backward compatibility if the structure changes. Current version is `SS3_STATUS_VERSION`.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3GetTime

Syntax:

`SS3_RETURN_CODE SS3GetTime(SSHANDLE` *`device, uint64_t *devicetime`*`)`

Description:

`SS3GetTime` retrieves the current time from the recorder in seconds since 00:00:00 Jan 1, 1970 UTC.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`devicetime` - pointer to a 64 bit integer to hold the time value.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3GetVersion

Syntax:

`SS3_RETURN_CODE SS3GetVersion( SSHANDLE` *`device`*`, PS_SS3_SWREV` *`version`*`, int structversion`)

Description:

`SS3GetVersion` gets the FPGA and firmware version information from the StreamStor device.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`version` – pointer to an S_SS3_SWREV structure to hold the version strings returned.

`structversion` - version of the structure to return. The latest version is `SS3_SWREV_VERSION`.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetApiVersion`

SS3GetWindowAddr

Syntax:

`SS3_RETURN_CODE SS3GetWindowAddr( SSHANDLE` *`device`*`, void*` *`address`*`)`

Description:

`SS3GetWindowAddr` returns the user virtual address of the recording data window. This address can be used to directly write data to the StreamStor device from a user program.

Note that the address range of this window can be retrieved using the `SS3GetBaseRange` function.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`. *`address`* – pointer to variable to hold address

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetBaseAddr`, `SS3GetBaseRange`

SS3IsWriteProtected

Syntax:

```
SS3_RETURN_CODE SS3IsWriteProtected( SSHANDLE device, LPCTSTR
recordingname, bool* protected)
```

Description:

`SS3IsWriteProtected` returns the write protection status of the specified recording.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.
`recordingname` - name of the recording to get write protect status.
`protected` - pointer to bool to return protection status (true indicates write protection is active)

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3NetOpen

Syntax:

```
SS3_RETURN_CODE SS3NetOpen( LPCTSTR address, uint16_t tcp_port,
SSHANDLE *device )
```

Description:

`SS3NetOpen` opens a StreamStor device over an Ethernet link and initializes the hardware and firmware in preparation for recording on an external interface. This function must be called before any other API function if using an Ethernet interface to StreamStor. After successful completion of this function, the handle pointed to by *`device`* can be used for all subsequent API calls.

NOTE: You should call `SS3Close` even if `SS3NetOpen` returns `SS3_FAIL`.

Parameters:

`address` - pointer to a string with a valid IPv4 address in dotted-quad notation (xxx.xxx.xxx.xxx), i.e. (“127.0.0.1”).

`tcp_port` - indicates which network TCP port the connection to StreamStor should use (default is 59427).

`device` - pointer to a StreamStor handle for initialization. Successful completion loads this parameter with a valid handle to the hardware device to use in subsequent API calls. *`*device`* is assigned the value `INVALID_SSHANDLE` on failure.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also: `SS3Close`

SS3Open

Syntax:

`SS3_RETURN_CODE SS3Open( uint32_t devIndex, SSHANDLE *pdevice )`

Description:

`SS3Open` opens a StreamStor device and initializes the hardware and firmware in preparation for recording. The device is transitioned to system ready state if required. This function must be called before any other API function. After successful completion of this function, the handle pointed to by `pdevice` can be used for all subsequent API calls.

NOTE: You should call `SS3Close` even if `SS3Open` returns `SS3_FAIL`.

Parameters:

`devIndex` - identifies the desired StreamStor to open when multiple StreamStor devices are in use. Use 1 for single card systems. Use `SS3DeviceFind` to find the number of devices installed.
`pdevice` - pointer to a StreamStor handle for initialization. Successful completion loads this parameter with a valid handle to the hardware device to use in subsequent API calls. `*pdevice` is assigned the value `INVALID_SSHANDLE` on failure.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3Close, SS3FindDevices`

SS3OpenRecording

Syntax:

`SS3_RETURN_CODE SS3OpenRecording( SSHANDLE` *`device`*`, LPCTSTR` *`name`*`, LPCTSTR` *`portname`*`, bool` *`forRead`* `)`

Description:

`SS3OpenRecording` will open an already existing recording for read or write. When a recording only has a single port recorded the `portname` parameter can be `NULL`. When only using a single recording and a single port both "`name`" and "`portname`" can be `NULL`. A recording can be opened for reading or writing but not both.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`name` - name of the recording to open.

`portname` - name of the recorded port to open.

`forRead` - determines if the recording will be opened for reading (`true`) or writing (`false`).

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3CloseRecording, SS3Write, SS3Read`

SS3Playback

Syntax:

```
SS3_RETURN_CODE SS3Playback(SSHANDLE device, LPCTSTR
recordingname)
```

Description:

`SS3Playback` puts StreamStor into playback mode where data is made available for transfer to an outside device. The playback will start from the beginning of the recording unless a play context has been supplied (SS3SetPlayContex). For external ports the playback will start immediately if the port is in the proper state for data flow to occur (flow control de-asserted, etc.).

Playback continues until:

- `SS3Stop` is called to halt the playback or
- all available data is played back

This function can be used for streaming data out an external port(s) or it can be used to allow a PCI Express device to source data from StreamStor.

If data was recorded on multiple ports:

- Data will playback on the stored port name. This can be modified by calling SS3BindOutputPort.
- You can specify non-zero starting points and playback lengths for each port with `SS3SetPlayContext`.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`recordingname` - name of the recording to be played.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns

SS3_FAIL.

Usage:

See Also:

SS3Stop, SS3SetPlayContext, SS3GetPlayLength

SS3PlaybackLoop

Syntax:

```
SS3_RETURN_CODE SS3PlaybackLoop( SSHANDLE device, LPCTSTR
recordingname, uint32_t loops )
```

Description

`SS3PlaybackLoop` starts data playback and loops back to the beginning when the end of data is reached. This continues until stop is called or the number of requested loops has been completed.

The `loops` parameter designates the number of times to play the entire data set before stopping. If `loops` is zero, the StreamStor will playback to the end of the data and then "loop" continuously. Playback will continue looping until `SS3Stop` is called.

To loop playback of subsets of a recording the `SS2SetPlayContext` can be used to define a start offset and length for the playback. The loop will play this subset according to the loops parameter provided. Note that the loop will always restart at the specified offset.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.
`recordingname` - name of the recording to be played.
`loops` - specifies the number of loops to perform before stopping. A value of 0 will loop continuously.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3Playback`

SS3Read

Syntax:

`SS3_RETURN_CODE SS3Read(SSHANDLE` *`device`*`, void*` *`bufaddr`*`, uint64_t` *`offset`*`, uint32_t *`*`length`*`)`

Description:

The `SS3Read` function reads data from the currently open recording. If the requested length exceeds the available data, the function will read to the end of the data and update the length parameter with the amount of data actually transferred. The supplied buffer must be within the range of the DMA buffer provided by the `SS3GetDMABufferAddr` and `SS3GetDMABufferSize`.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.
`bufaddr` - address within the DMA memory buffer. See `SS3GetDMABufferAddr`.
`address` - offset into the recorded data set where the data should start reading.
`length` – pointer to an integer with the number of bytes requested to read from the recording into the buffer. The buffer must be at least this same size. On successful completion this parameter is changed to indicate the number of bytes transferred.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3OpenRecording, SS3CloseRecording`

SS3Record

Syntax:

SS3_RETURN_CODE SS3Record(SSHANDLE *device*, LPCTSTR *recordingname*, bool *wrapenable*)

Description:

SS3Record starts the record mode of the StreamStor device. After a successful call of this function, the StreamStor device will record to the currently open recording any data coming to the input port(s) as defined when the recording was created. Recording will continue until the storage space is full, the size defined by the file is reached or until SS3Stop is called (whichever occurs first). Note that a non-empty recording must be erased first with SS3EraseRecording.

Parameters:

device - device handle returned from a previous call to SS3Open.

recordingname - name of the recording to use.

wrapenable - not supported, use false (0)

Return Value:

On success, this function returns SS3_SUCCESS.

On failure, this function returns SS3_FAIL.

Usage:

See Also:

SS3Stop

SS3RenameRecording

Syntax:

`SS3_RETURN_CODE SS3RenameRecording( SSHANDLE` *`device`*`, LPCTSTR` *`currentname`*`, LPCTSTR` *`newname`* `)`

Description:

`SS3RenameRecording` changes the name of the specified recording. This allows the user to change the name of an existing recording.

Note that if you call `SS3EraseRecording` to erase a recording, the name is retained.

If a recording is write protected this function will fail.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.
`currentname` - name of existing recording.
`newname` - new name to use for recording.

Note that the new name can be no more than `SS3_MAX_RECORDINGNAME` characters in length.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetRecordingName, SS3CreateRecording, SS3EraseRecording`

SS3SetPlayContext

Syntax:

`SS3_RETURN_CODE SS3SetPlayContext(SSHANDLE` *`device`*`, LPCTSTR` *`recordingname`*`, LPCTSTR` *`portname`*`, uint64_t` *`offset`*`, uint64_t` *`length`*`)`

Description:

`SS3SetPlayContext` defines a non-standard play context for the specified recording and port. This allows a playback to be started from an offset into the recording and / or to specify a playback length to be specified other than the entire recording. The play context settings are used by both `SS3Playback` and `SS3PlaybackLoop` functions.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.
`recordingname` - name of recording.
`portname` - name of recorded port. May be NULL if only one port is recorded.
`offset` - offset into the recording to use for playback start.
`length` - length of playback (per loop if using `SS3PlaybackLoop`).

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3Playback, SS3PlaybackLoop`

SS3SetRecorderName

Syntax:

`SS3_RETURN_CODE SS3SetRecorderName(SSHANDLE` *`device`*`, LPCTSTR` *`recordername`* `)`

Description:

`SS3SetRecorderName` assigns a name to the recorder.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.
`recordername` - new name for the recorder.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3GetRecorderName`

SS3SetTime

Syntax:

`SS3_RETURN_CODE SS3SetTime(SSHANDLE` *`device`*`, uint64_t` *`devicetime`*`)`

Description:

`SS3SetTime` sets the current time on the recorder in seconds since `00:00:00 Jan 1, 1970 UTC.` The time is used to timestamp recordings.

Parameters:

`device` - device handle returned from a previous call to `SS3Open.`
`devicetime` - time value.

Return Value:

On success, this function returns `SS3_SUCCESS.`
On failure, this function returns `SS3_FAIL.`

Usage:

See Also:

SS3SetWriteProtect

Syntax:

`SS3_RETURN_CODE SS3SetWriteProtect( SSHANDLE` *`device`*`, LPCTSTR` *`recordingname`* `)`

Description:

`SS3SetWriteProtect` marks the specified recording as write protected. After write protection is set, subsequent attempts to alter the recorded data (i.e., calls to `SS3Record`, or `SS3EraseRecording)` in that recording will return an error.

Note that this protection is only honored by the StreamStor software and does not prevent erasure by other software or techniques.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.
`recordingname` - name of recording to write protect.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3ClearWriteProtect`

SS3Stop

Syntax:

```
SS3_RETURN_CODE SS3Stop( SSHANDLE device )
```

Description:

`SS3Stop` will halt a recording operation and make sure all data is flushed to disk. This function should always be used to end a recording even when all disk space has been exhausted.

`SS3Stop` is also used to halt a playback initiated by `SS3Playback / SS3PlaybackLoop`.

If the StreamStor is recording or playing on multiple ports, calling `SS3Stop` stops recording (or playback) on all ports.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3Record, SS3Playback`

SS3TruncateRecording

Syntax:

`SS3_RETURN_CODE SS3TruncateRecording(SSHANDLE` *`device`*`, LPCTSTR` *`recordingname`*`)`

Description:

`SS3TruncateRecording` can be used to reduce the file size of an existing recording. This function is typically used when the "0" length option was used when creating a recording and now needs to be truncated to allow additional recordings to be recorded. This function will reduce the amount of storage space used by a recording to the smallest possible without deleting any recorded data.

Note that all ports in a multi-port recording will be truncated.

Parameters:

`device` - is the device handle returned from a previous call to `SS3Open`.
`recordingname` - is the name of the recording to truncate.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3UpdateHardware

Syntax:

`SS3_RETURN_CODE SS3UpdateHardware(SSHANDLE` *`device`*`, LPCSTR` *`flashpath`*`)`

Description:

`SS3UpdateHardware` is used to update the firmware on the Cobra board. This function should only be used at the direction of Conduant support. Attempting to flash with an invalid file can render your system inoperable.

Parameters:

`device` - is the device handle returned from a previous call to `SS3Open`.
`flashpath` - is the path name of the flash file to be used for the update.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3UploadSystemFile

Syntax:

`SS3_RETURN_CODE SS3UploadSystemFile( SSHANDLE` *`device`*`, LPCTSTR` *`filename`* `)`

Description:

`SS3UploadSystemFile` is used to update the software on the Cobra system. This function should only be used at the direction of Conduant support. Uploading an invalid file can render your system inoperable.

Parameters:

`device` - is the device handle returned from a previous call to `SS3Open`.
`filename` - is the path name of the file to be uploaded.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3Write

Syntax:

```
SS3_RETURN_CODE SS3Write( SSHANDLE device, void* bufaddr, uint32_t
*length)
```

Description:

SS3Write writes data from a user memory buffer to the open recording. Data is appended to any existing data in the recording. Applications should always check the returned length value. If the system cannot fit the entire length in the space available in the recording, the return length value will indicate how much data was written.

Parameters:

device - device handle returned from a previous call to SS3Open.
bufaddr - pointer to a buffer holding the data to write to the recording.
length - number of bytes to write from the user buffer.

Return Value:

On success, this function returns SS3_SUCCESS.
On failure, this function returns SS3_FAIL.

Usage:

See Also:

“C” Structures

This section contains descriptions of the structures used by “C” functions.

Structure S_SS3_DEVINFO

```
typedef struct _DEVINFO
{
   wchar_t BoardType[SS3_MAX_NAME];
   uint32_t SerialNum;
   uint32_t NumDrives;
   uint32_t NumExtPorts;
   uint64_t TotalCapacityBytes;
   uint64_t FreeCapacityBytes;
}S_SS3_DEVINFO, *PS_SS3_DEVINFO;
```

Purpose:

This structure is used by the `SS3GetDeviceInfo` function to return information about the StreamStor system configuration.

Members:

`BoardType` - board type (model name).

`SerialNum` - serial number of the StreamStor board.

`NumDrives` - number of drives currently connected and configured on the StreamStor recorder.

`NumExtPorts` – number of external ports. Does not include ports used for PCI Express or Ethernet.

`TotalCapacityBytes` – Total capacity of the recorder, expressed in bytes.

`FreeCapacityBytes` – Free (unused) capacity of the recorder, expressed in bytes.

Structure S_SS3_STATUS

```
typedef struct _SS3_STATUS
{
   bool Ready;
   bool Recording;
   bool Playing;
   bool RecordingOpen;
   bool BufferOverflow;
   bool StorageOverflow;
   bool UnitAttention;
   bool SysErrorOccurred;
   uint32_t SysErrorCode;
}S_SS3_STATUS, *PS_SS3_STATUS;
```

Purpose:

This structure holds various system status flags as returned by the `SS3GetStatus` function.

Members:

`Ready` – System ready flag, indicates the system firmware and hardware have been initialized successfully.

`Recording` – Indicates that the system is currently in a record mode.

`Playing` – Indicates that the system is currently in a playback mode.

`RecordingOpen` – A recording is currently open for reading or writing.

`BufferOverflow` – The device experienced a buffer overflow and data was likely lost.

`StorageOverflow` – The device experienced a storage overflow and data recording has ended.

`UnitAttention` – Indicates that the system has experienced a power cycle or reset since it was opened. Software should consider this a failure condition.

`SysErrorOccurred` – Indicates that a system initialization failure or other serious issue has occurred.

`SysErrorCode` – Holds error code if `SysErrorOccurred` is TRUE.

Structure S_SS3_DRIVEINFO

```
typedef struct _SS3_DRIVEINFO
{
   wchar_t Model[SS3_MAX_DRIVENAME];
   wchar_t Serial[SS3_MAX_DRIVESERIAL];
   wchar_t Vendor[SS3_MAX_DRIVEVENDOR];
   uint64_t Capacity;
}S_SS3_DRIVEINFO, *PS_SS3_DRIVEINFO;
```

Purpose:

This structure is used by the `SS3GetDriveInfo` function to return information about the disk drives on the StreamStor system.

Members:

`Model` – Drive model name
`Serial` – Drive serial number
`Capacity` – Drive capacity (bytes)

Structure S_PORT_FILE

```
typedef struct
{
   char portname[SS3_MAX_NAME];
   uint64_t recordingLength;
}S_PORT_FILE
```

Purpose:

This structure is used by `SS3CreateMultiPortRecording` to allow the definition of an arrays of ports to use when recording..

Members:

`portname` – Name of the port to use when recording.
`recordingLength` – Length in bytes of the recording file to create for this port.

Structure S_SS3_SWREV

```
typedef struct _SS3_SWREV
{
   wchar_t FirmwareVersion[SS3_VERSION_LENGTH];
   wchar_t FirmDateCode[SS3_DATECODE_LENGTH];
   wchar_t FPGAVersion[SS3_VERSION_LENGTH];
   wchar_t DriverVersion[SS3_VERSION_LENGTH];
}S_SS3_SWREV, *PS_SS3_SWREV;
```

Purpose:

This structure is used by `SS3GetVersion` to return software/hardware version strings.

Members:

`FirmwareVersion` – StreamStor firmware version.
`FirmDateCode` – Build date of the firmware.
`FPGAVersion` – FPGA Controller logic version.
`DriverVersion` – Driver version.

“C” ODI Functions

This section contains functions specific to the ODI optional recorder implementation that provides Interlaken packet recording and playback capabilities. The functions in this section perform all the management required to organize data into Interlaken packets. The software implements a multiport recorder where one port is the packet data and a second "virtual" port is used to capture the packet boundaries and errors.

When a recorder is opened with one of the device open functions (`SS3_OdiOpen`, `SS3_OdiNetOpen`) in this section, only the other functions in this section can be used to control the recorder. The exception to this is that if a function does not have an `SSHANDLE` parameter (e.g. `SS3GetApiVersion`), then it can still be used normally.

SS3_OdiClose

Syntax:

`void SS3_OdiClose( SSHANDLE device )`

Description:

SS3_OdiClose closes the open StreamStor device. This should be called before exiting an application that has opened a StreamStor device with SS3_OdiOpen or another Odi open function. No other application can open the StreamStor device until this function has been called.

Parameters:

device - device handle returned from a previous Odi open call.

Return Value:

None.

Usage:

See Also:

SS3_OdiOpen, SS3_OdiNetOpen

SS3_OdiCloseRecording

Syntax:

`SS3_RETURN_CODE SS3_OdiCloseRecording( SSHANDLE` *`device`* `)`

Description:

`SS3_OdiCloseRecording` will close an open recording. After any operations(s) to read or write data to a recording, the recording must be closed using this function before any other activity can occur.

Parameters:

`device` - device handle returned from a previous call to SS3_OdiOpen.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3_OdiOpenRecording`

SS3_OdiCreateRecording

Syntax:

`SS3_RETURN_CODE SS3_OdiCreateRecording( SSHANDLE` *`device`*`, LPCTSTR` *`recordingname`*`, uint64_t` *`size`* `)`

Description:

`SS3_OdiCreateRecording` will create a new recording on the system in preparation for recording (or writing) data. If the size parameter is 0, the system will create the largest possible recording.

Parameters:

`device` - device handle returned from a previous call to `SS3_OdiOpen`.
`recordingname` - name to assign to the new recording.
`size` - size (bytes) of the recording to create.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3_OdiDeleteRecording

Syntax:

`SS3_RETURN_CODE SS3_OdiDeleteRecording( SSHANDLE` *`device`*`, LPCTSTR` *`recordingname`* `)`

Description:

`SS3_OdiDeleteRecording` will delete an existing recording from the system. If the recording is the only one on the system it will not be removed completely but will be erased and renamed to system default name.

Parameters:

`device` - device handle returned from a previous call to `SS3_OdiOpen`.
`recordingname` - name of the recording to delete.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3_OdiDownloadRecording

Syntax:

`SS3_RETURN_CODE SS3DownloadRecording( SSHANDLE device, LPCTSTR recordingname, LPCTSTR filepath )`

Description:

`SS3DownloadRecording` can be used to download recorded data directly to a file on the local system.

Parameters:

`device` - device handle returned from a previous call to `SS3_OdiOpen`.

`recordingname` - name of the recording to download.

`filepath` - file to write with downloaded data.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3_OdiEraseRecording

Syntax:

`SS3_RETURN_CODE SS3_OdiEraseRecording( SSHANDLE` *`device`*`, LPCTSTR` *`recordingname`* `)`

Description:

`SS3_OdiEraseRecording` will erase any existing data in the named recording. The recording name is not affected. A recording must be erased or empty before it can be used in a new recording.

Parameters:

`device` - device handle returned from a previous call to `SS3_OdiOpen`.
`recordingname` - name of the recording to erase.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3_OdiGetDeviceInfo

Syntax: `SS3_RETURN_CODE SS3GetDeviceInfo( SSHANDLE device, S_SS3_DEVINFO *devInfo, int structversion )`

Description:

`SS3GetDeviceInfo` retrieves information from the StreamStor device about its fixed configuration. The function will use the version of the structure supplied in the *`structversion`* parameter to allow backward compatibility if the structure changes.

Parameters:

`device` - device handle returned from a previous call to `SS3_OdiOpen`.

`pDevInfo` - pointer to an `S_SS3_DEVINFO` structure.

`structversion` - version number of the structure to return. Current version is `SS3_DEVINFO_VERSION`.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3_OdiGetDriveInfo

Syntax:

```
SS3_RETURN_CODE SS3_OdiGetDriveInfo( SSHANDLE device, int drive,
S_SS3_DRIVEINFO* pDriveInfo, int structversion )
```

Description:

`SS3_OdiGetDriveInfo` retrieves info from the StreamStor recorder about a specific drive.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.
`drive` - index number of the drive, starts at 0 and must be less the count reported using `SS3_OdiGetDeviceInfo` (`S_SS3_DEVINFO.NumDrives`).
`pDriveInfo` - pointer to an `S_SS3_DRIVEINFO` structure to hold returned information.
`structversion` - version of structure to return. Current version is `SS3_DRIVEINFO_VERSION`.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3_OdiGetDeviceInfo`

SS3_OdiGetFreeCapacity

Syntax:

`SS3_RETURN_CODE SS3_OdiGetFreeCapacity( SSHANDLE` *`device`*`, uint64_t *`*`capacity` `)`

Description:

`SS3_OdiGetFreeCapacity` retrieves the total amount of free space left on the recording drives. The amount available for recording will be smaller due to overhead used to manage files and store packet boundaries.

Parameters:

`device` - device handle returned from a previous call to `SS3_OdiOpen`.
`capacity` - pointer to a 64 bit integer to hold the free space (bytes).

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3_OdiGetLength

Syntax:

```
SS3_RETURN_CODE SS3_OdiGetLength( SSHANDLE device, LPCTSTR
recordingname, uint64_t *length )
```

Description:

`SS3_OdiGetLength` retrieves the total amount of data in a recording. This is sum of all the data in every packet but does not include the size of any metadata captured for managing packet boundaries.

Parameters:

`device` - device handle returned from a previous call to `SS3_OdiOpen`.
`recordingname` - name of the recording to query.
`length` - pointer to a 64 bit integer to hold the data length.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3_OdiGetPacketCount

Syntax:

`SS3_RETURN_CODE SS3_OdiGetPacketCount(SSHANDLE *device*, LPCTSTR *recordingname*, uint64_t **count*)`

Description:

`SS3_OdiGetPacketCount` returns the number of packets in the specified recording.

Parameters:

`device` - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
`recordingname` - name of the recording
`count` - pointer to 64 bit integer to hold the number of packets recorded.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

SS3_OdiGetPacketErrorCount

Syntax:

`SS3_RETURN_CODE SS3_OdiGetPacketErrorCount(SSHANDLE` *`device`*`, uint32_t *`*`errcount`*`)`

Description:

`SS3_OdiGetPacketErrorCount` reports the number of packet errors encountered by the StreamStor during the most recent Odi recording session. This count is not preserved after a device reset.

Parameters:

`device` - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
`errcount` - pointer to an integer to hold the error count.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

SS3_OdiGetPlayLength

Syntax:

`SS3_RETURN_CODE SS3_OdiGetPlayLength( SSHANDLE` *`device`*`, LPCTSTR` *`recordingname`*`, uint64_t*` *`length`*`)`

Description:

`SS3_OdiGetPlayLength` returns the length (in bytes) of the most recent playback of a recording in a 64-bit integer. The play length is volatile and is only available within the same session as a playback occurred.

Parameters:

`device` - handle provided by `SS3_OdiOpen or SS3OdiNetOpen`.
`recordingname` - name of the recording
`length` - pointer to parameter to hold playback length.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3_OdiGetProgress

Syntax:

`SS3_RETURN_CODE SS3_OdiGetProgress(SSHANDLE` *`device`*`, uint64_t *`*`progress`*`)`

Description:

`SS3_OdiGetProgress` reports the amount of data transferred while a recording or playback is in progress. Note that the value returned will not be accurate since the amount is continuously changing. The value can be used as a reasonable approximation of progress to provide feedback.

Parameters:

`device` - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
`progress` - pointer to a 64 bit integer to hold the amount of data transferred.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

SS3_OdiGetRecorderName

Syntax:

```
SS3_RETURN_CODE SS3_OdiGetRecorderName( SSHANDLE device,
LPTSTR recordername, int *strsize )
```

Description:

This function retrieves the current name of the recorder. A user defined name can be assigned with `SS3_OdiSetRecorderName`. A default system name is used when a custom name has not been assigned. If no string pointer is provided to return the name, the size of the string required will be retrieved.

Parameters:

`device` - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
`recordername` - string to hold the recorder name.
`strsize` - pointer to integer with size of the above string in number of characters. If *`recordername`* is NULL this parameter will be set to the size of the string required.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

SS3_OdiGetRecordingCount

Syntax:

```
SS3_RETURN_CODE SS3_OdiGetRecordingCount( SSHANDLE device, int
*count )
```

Description:

This function retrieves the number of recordings on the system.

Parameters:

device - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
count - pointer to integer to hold count of recordings.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

SS3_OdiGetRecordingFileSize

Syntax:

`SS3_RETURN_CODE SS3_OdiGetRecordingFileSize(SSHANDLE` *`device`*`, LPTCSTR` *`recordingname`*`, uint64_t` *`*size`*`)`

Description:

`SS3_OdiGetRecordingFileSize` provides a way to retrieve the amount of storage space (bytes) allocated for the recording data. The amount returned does not include the amount automatically allocated for storing packet sizes.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.

`recordingname` - name of the recording.

`size` – pointer to variable to hold returned file size.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3_OdiTruncateRecording, SS3_OdiCreateRecording`

SS3_OdiGetRecordingName

Syntax:

```
SS3_RETURN_CODE SS3_OdiGetRecordingName( SSHANDLE device,
LPTSTR recordingname, int *strsize, int index )
```

Description:

This function retrieves the name of a recording. If no string pointer is provided to return the name, the size (number of characters) of the string required will be retrieved.

Parameters:

device - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
recordingname - string to hold the recording name.
strsize - pointer to integer with size of string in characters. If string is NULL, this parameter will be set to the size of the string required (in characters).
index - index of recording name to be retrieved. Must be less than the number of recordings returned by `SS3_OdiGetRecordingCount`.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

SS3_OdiGetStatus

Syntax:

```
SS3_RETURN_CODE SS3_OdiGetStatus( SSHANDLE device, S_SS3_STATUS
*devStatus, int structversion )
```

Description:

`SS3_OdiGetStatus` retrieves the current status of the StreamStor device.

Parameters:

`device` - device handle returned from a previous call to `SS3Open`.
`devStatus` - pointer to an `S_SS3_STATUS` structure.
`structversion` - version of the structure to retrieve for backward compatibility if the structure changes. Current version is `SS3_STATUS_VERSION`.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3_OdiGetTotalCapacity

Syntax:

```
SS3_RETURN_CODE SS3_OdiGetTotalCapacity( SSHANDLE device,
uint64_t *capacity )
```

Description:

This function retrieves the total capacity available on the recorder. This includes all space already utilized for existing recordings.

Parameters:

device - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
capacity - pointer to a 64 bit integer to hold the capacity of the system in bytes.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

SS3_OdiGetVersion

Syntax: `SS3_RETURN_CODE SS3GetVersion( SSHANDLE device, PS_SS3_SWREV version, int structversion )`

Description: `SS3_OdiGetVersion` gets the FPGA and firmware version information from the StreamStor device.

Parameters:

`device` - device handle returned from a previous call to `SS3_OdiOpen` or `SS3_OdiNetOpen`.
`version` - pointer to an S_SS3_SWREV structure to hold the version strings returned.
`structversion` - version of the structure to return. The latest version is `SS3_SWREV_VERSION`.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also: `SS3GetApiVersion`

SS3_OdiNetOpen

Syntax:

```
int SS3_OdiNetOpen( LPCTSTR address, uint16_t tcpport,
SSHANDLE *device )
```

Description:

`SS3_OdiNetOpen` opens a StreamStor device over an Ethernet link and initializes the hardware and firmware in preparation for recording on an external interface. This function must be called before any other API function if using an Ethernet interface to StreamStor. After successful completion of this function, the handle pointed to by device can be used for all subsequent API calls.
This function is intended for use when opening Odi recorders to allow the use of other Odi functions as described in this section.

Parameters:

`address` - pointer to a string with a valid IPv4 address in dotted-quad notation (xxx.xxx.xxx.xxx), i.e. (“127.0.0.1”).

`tcpport` - indicates which TCP network port the connection to StreamStor should use (default is 59427).

`device` - pointer to a StreamStor handle for initialization. Successful completion loads this parameter with a valid handle to the hardware device to use in subsequent API calls. *`device` is assigned the value `INVALID_SSHANDLE` on failure.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3_OdiClose`

SS3_OdiOpen

Syntax:

`int SS3_OdiOpen(uint32_t devindex, SSHANDLE *device)`

Description:

`SS3_OdiOpen` opens a StreamStor device and initializes the hardware and firmware in preparation for recording. The device is transitioned to system ready state if required. This function must be called before any other API function. After successful completion of this function, the handle pointed to by device can be used for all subsequent API calls.

Parameters:

`devindex` - index number of the StreamStor device connected by PCI Express to open. The device must be an Odi type recorder.

**`device`* - pointer to an `SSHANDLE` type that will be populated with a valid handle to use when calling other functions.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3_OdiClose`, `SS3_OdiNetOpen`

SS3_OdiOpenRecording

Syntax:

```
SS3_RETURN_CODE SS3_OdiOpenRecording( SSHANDLE device, LPCTSTR recordingname, bool forRead )
```

Description:

`SS3_OdiOpenRecording` will open an already existing recording for read or write. When reading or writing data is completed the open recording should be closed with `SS3_OdiCloseRecording`.

Parameters:

`device` - device handle returned from a previous call to SS3_OdiOpen.
`name` - name of the recording to open.
`forRead` - should be true to open the recording for reading and false to open for writing

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3_OdiCloseRecording`, `SS3_OdiWritePacket`, `SS3_OdiReadPacket`

SS3_OdiPlayback

Syntax:

```
SS3_RETURN_CODE SS3_OdiPlayback( SSHANDLE device, LPCTSTR
recordingname, uint32_t loops )
```

Description:

`SS3_OdiPlayback` will begin playback of an existing recording. The playback can be looped by providing a "loops" number larger than one (1). A "loops" parameter of zero (0) will loop continuously until `SS3_OdiStop` is called.

Parameters:

`device` - device handle returned from a previous call to `SS3_OdiOpen`.
`recordingname` - name of the recording to playback.
`loops` - number of times the recording should be played back.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3_OdiStop`

SS3_OdiReadPacket

Syntax:

```
SS3_RETURN_CODE SS3_OdiReadPacket(SSHANDLE device, uint64_t
packet, void *buffer, uint32_t *size)
```

Description:

`SS3_OdiReadPacket` reads a single Odi packet from the StreamStor recorder. The specific recording must be opened with `SS3OpenRecording` (for read) prior to this call. Note that the StreamStor Odi recorder is designed for optimum performance when packets are read in sequential order. The data that is returned is the contents of the Interlaken/Odi packet with no Interlaken headers. Note that the retrieved packet size is written to the size variable on successful completion.

Parameters:

`device` - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
`packet` - index number to specify the packet to be retrieved. Packet indexing starts at 1 for the first packet in a recording.
`buffer` - pointer to the memory space to hold the packet. If the buffer is not large enough to hold the requested packet an error will be returned and the data in the buffer will not be overwritten.
`size` - pointer to an integer that provides the size in bytes of the memory area defined by buffer. This value is written with the packet size returned in the buffer.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3_OdiOpenRecording`, `SS3_OdiWritePacket`

SS3_OdiRecord

Syntax:

```
SS3_RETURN_CODE SS3_OdiRecord( SSHANDLE device, LPCTSTR
recordingname, bool enablewrap )
```

Description:

`SS3_OdiRecord` initiates recording of data from the Interlaken port. Recording will continue until the end of available storage space in the recording is exhausted. If "`enablewrap`" is true, the recording will continue indefinitely by overwriting the oldest data. The recording will use all available storage space before overwriting old data. When "`enablewrap`" is true, the recording must be ended by eventually calling `SS3_OdiStop`.

Parameters:

`device` - device handle returned from a previous call to `SS3_OdiOpen`.
`recordingname` - name of the recording to playback.
`enablewrap` - enable wrap recording (circular).
`maxsize` - amount of storage space to use before ending or wrapping the recording.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3_OdiStop`

SS3_OdiRenameRecording

Syntax:

```
SS3_RETURN_CODE SS3_OdiRenameRecording( SSHANDLE device, LPCTSTR
currentname, LPCTSTR newname )
```

Description:

`SS3_OdiRenameRecording` changes the name of the specified recording. This allows the user to change the name of an existing recording. Note that if you call `SS3_OdiEraseRecording` to erase a recording, the name is retained. If a recording is write protected this function will fail.

Parameters:

`device` - device handle returned from a previous call to `SS3_OdiOpen`.
`currentname` - name of existing recording.
`newname` - new name to use for recording. Note that `newname` can be no more than `SS3_MAX_RECORDINGNAME` characters.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3_OdiGetRecordingName`, `SS3_OdiCreateRecording`, `SS3_OdiEraseRecording`

SS3_OdiSetRecorderName

Syntax:

```
SS3_RETURN_CODE SS3_OdiSetRecorderName( SSHANDLE device,
LPCTSTR recordername )
```

Description:

This function sets the name of the recorder. A user defined name can be assigned with this function and retrieved with `SS3_OdiGetRecorderName`. A default system name is used when a custom name has not been assigned.

Parameters:

`device` - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
`recordername` - string to hold the recorder name.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3_OdiGetRecorderName`

SS3_OdiStop

Syntax:

`SS3_RETURN_CODE SS3_OdiStop( SSHANDLE` *`device`* `)`

Description:

`SS3_OdiStop` will stop any recording or playback operation in progress.

Parameters:

`device` - device handle returned from a previous call to `SS3_OdiOpen`.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3_OdiRecord`, `SS3OdiPlayback`

SS3_OdiTruncateRecording

Syntax:

```
SS3_RETURN_CODE SS3TruncateRecording(SSHANDLE device, LPTCSTR recordingname)
```

Description:

`SS3_OdiTruncateRecording` can be used to reduce the file size of an existing recording. This function is typically used when the "0" length option was used when creating a recording and now needs to be truncated to allow additional recordings to be recorded. This function will reduce the amount of storage space used by a recording to the smallest possible without deleting any recorded data.

Parameters:

`device` - is the device handle returned from a previous call to `SS3Open`.
`recordingname` - is the name of the recording to truncate.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3_OdiGetRecordingFileSize`

SS3_OdiUploadRecording

Syntax:

`SS3_RETURN_CODE SS3UploadRecording( SSHANDLE` *`device`*`, LPCTSTR` *`recordingname`*`, LPCTSTR` *`filepath`*`, uint32_t` *`packetSize`* `)`

Description:

`SS3UploadRecording` can be used to upload recorded data directly from a file on the local system to a recording. The recording file size must be large enough to completely hold the recording. This function only supports recordings that will utilize fixed size packets. The local file size must be an even multiple of the packet size provided. Note that packet sizes must be a minimum of 64 bytes, a maximum of 262,144 bytes and must be a multiple of 32.

Parameters:

`device` - device handle returned from a previous call to SS3_OdiOpen.
`recording name` - name of the recording to download.
`filepath` - file to write with downloaded data.
`packetSize` - size of packet to use when creating the recording.

Return Value:

On success, this function returns `SS3_SUCCESS`. On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3_OdiUploadSystemFile

Syntax:

`SS3_RETURN_CODE SS3_OdiUploadSystemFile( SSHANDLE` *`device`*`, LPCSTR` *`filename`* `)`

Description:

`SS3_OdiUploadSystemFile` is used to update the software on the Cobra system. This function should only be used at the direction of Conduant support. Uploading an invalid file can render your system inoperable.

Parameters:

`device` - is the device handle returned from a previous call to `SS3Open`.
`filename` - is the path name of the file to be uploaded.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

SS3_OdiWritePacket

Syntax:

```
SS3_RETURN_CODE SS3_OdiWritePacket(SSHANDLE device, void
*buffer, uint32_t *size)
```

Description:

`SS3_OdiWritePacket` writes a single Odi packet to the StreamStor recorder. The specific recording must be opened with `SS3_OdiOpenRecording` (for write) prior to this call. Packets must be written in the desired order. Any packets written are added to the recording in the order received and appended to any existing data. Applications should always check the size returned from this function. If the recording has no more space, the returned size will be 0.

Parameters:

`device` - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
`buffer` - pointer to the memory space holding the packet data.
`size` - number of bytes in the packet.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

See Also:

`SS3CloseRecording`, `SS3_OdiWritePacket`, `SS3_OdiReadPacket`

“C” Odi Port Functions

This section contains functions specific to the Odi port as defined by the AXIe consortium specification for Odi.

SS3_OdiPortActivate

Syntax:

```
SS3_RETURN_CODE SS3_OdiPortActivate(SSHANDLE device, LPCTSTR port,
   LaneRate laneRate, int txBurstMax, FlowControl flowControl,
   LPCTSTR options)
```

Description:

`SS3_OdiPortActivate` enables the specified Odi port. When enabled the port will transmit and receive on the Odi Interlaken interface.

Parameters:

`device` - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
`port` - name of the port to activate.
`lanerate` - lane rate (Gbps) to use when the port is enabled. See LaneRate enum definition for valid values.
`txBurstMax` - maximum transmit burst that should be used by the port.
`flowControl` - type of Flow Control that should be used by the port. See FlowControl enumeration for valid values.
`options` - not currently used and can be NULL

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

SS3_OdiPortCount

Syntax:

`SS3_RETURN_CODE SS3_OdiPortCount(SSHANDLE *device*, int **count*)`

Description:

`SS3_OdiPortCount` returns the number of Odi ports available in the system.

Parameters:

`device` - handle provided by SS3_OdiOpen or SS3_OdiNetOpen.
`count` - pointer to integer to hold number of Odi ports available.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

SS3_OdiPortDeactivate

Syntax:

```
SS3_RETURN_CODE SS3_OdiPortDeactivate(SSHANDLE device, LPCTSTR
   port)
```

Description:

`SS3_OdiPortDeactivate` disables the specified Odi port.

Parameters:

`device` - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
`port` - name of the port to activate.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

SS3_OdiPortName

Syntax:

```
SS3_RETURN_CODE SS3_OdiPortName(SSHANDLE device, LPTSTR name, int
   size, int index)
```

Description:

`SS3_OdiPortName` returns the name of an Odi port.

Parameters:

`device` - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
`name` - user provided string buffer to hold the name.
`size` - number of characters available in the name string.
`index` - 0 based index of the port name to return.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

SS3_OdiPortGetCapability

Syntax:

```
SS3_RETURN_CODE SS3_OdiPortGetCapability(SSHANDLE device, LPCTSTR
   port, int structversion, S_PORT_CAPABILITY *capability)
```

Description:

`SS3_OdiPortGetCapability` retrieves the capability structure for the specified port as defined by the Odi standard.

Parameters:

`device` - handle provided by SS3_OdiOpen or SS3_OdiNetOpen.
`port` - name of the port
`structversion` - version of the `S_PORT_CAPABILITY` structure to return. Applications should normally use `S_PORT_CAPABILITY_VERSION` in this parameter.
`capability` - structure to return.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

SS3_OdiPortGetStatus

Syntax:

```
SS3_RETURN_CODE SS3_OdiPortGetStatus(SSHANDLE device, LPCTSTR
   port, uint32_t *status)
```

Description:

`SS3_OdiPortGetStatus` retrieves status information for the specified port as defined by the Odi standard. Replace "X" below with "B" for a bit number or with "M" for a value mask. (e.g. `SS3_B_PORT_TX_READY` = 1, `SS3_M_PORT_TX_READY` = 0x00000002)
Status definitions:

- `SS3_X_PORT_ACTIVE` - Indicates whether the port has been activated.
- `SS3_X_PORT_TX_READY` - Indicates if the port is ready to transmit.
- `SS3_X_PORT_RX_READY` - Indicates if the port is ready to receive.
- `SS3_X_PORT_RX_LANE_ERROR` - Indicates if the port has encountered a lane error.
- `SS3_X_PORT_RX_BURST_MAX_ERROR` - Indicates it the port has encountered a burst maximum size violation.
- `SS3_X_PORT_CRC_ERROR` - Indicates if the port has encountered a CRC data error.
- `SS3_X_PORT_RX_OVERRUN` - Indicates if the port has had a receive buffer overrun (overflow).
- `SS3_X_PORT_FC_STATUS` - Indicates if the port is currently flow controlled.

Parameters:

`device` - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
`port` - name of the port
`status` - pointer to a 32 bit value to store the bit significant status.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

SS3_OdiPortGetStatistics

Syntax:

```
SS3_RETURN_CODE SS3_OdiPortGetStatistics(SSHANDLE device, LPCTSTR
   port, int structversion, S_PORT_STATISTICS *statistics)
```

Description:

`SS3_OdiPortGetStatistics` retrieves statistical information for the specified port as defined by the Odi standard.

Parameters:

`device` - handle provided by `SS3_OdiOpen` or `SS3_OdiNetOpen`.
`port` - name of the port
`structversion` - version of the `S_PORT_STATISTICS` structure to return. Applications should normally use `S_PORT_STATISTICS_VERSION` in this parameter.
`statistics` - pointer to the structure to be filled.

Return Value:

On success, this function returns `SS3_SUCCESS`.
On failure, this function returns `SS3_FAIL`.

Usage:

“C” OdiPort Structures

This section contains definitions of structures and enumerations used by the preceding Odi functions. These structures are defined according to the Odi specification of the AXIe consortium.

Structure S_PORT_CAPABILITY

```
typedef struct
{
   char name[64];
   char version[64];
   enum LaneRate laneRates[8]; // 0 terminated list
   uint32_t lanes[8]; // 0 terminated list
   uint32_t txBurstMaxes[8]; // 0 terminated list
   uint32_t rxBurstMax;
   enum FlowControl flowControls[32]; // 0 terminated list
   uint32_t channelMax;
   bool txRateMatching;
}S_PORT_CAPABILITY;
```

Purpose:

This structure is used by `SS3_OdiPortGetCapability` to return information about the capabilities of a port. The constant `S_PORT_CAPABILITY_VERSION` is always the latest version number of the structure.

Members

`name` – The port name string.
`version` – Version of the port implementation
`laneRates` – Array of lane rates supported
`lanes` – Number of lanes used by this port.
`txBurstMaxes` – Array of TX burst max sizes supported
`rxBurstMax` – The size of RX burst max the port will use
`flowControls` – Arrays of flow control types supported
`channelMax` – Maximum number of Interlaked channels supported
`txRateMatching` – Indicates if port supports TX rate control

Structure S_PORT_STATISTICS

```
typedef struct
{
   uint64_t BytesSent;
   uint64_t BytesReceived;
   uint64_t BadBurstsReceived;
   uint64_t TxFlowControlHoldoffs;
}S_PORT_STATISTICS;
```

Purpose

This structure is used by `SS3_OdiPortGetStatistics` to return information about the operation of a port. The constant `S_PORT_STATICTICS_VERSION` is always the latest version number of the structure.

Members

`BytesSent` –

`BytesReceived` –

`BadBurstsReceive` –

`TxFlowControlHoldoffs` –

enum FlowControl

```
enum FlowControl
{
   NONE = 1,
   INBAND = 2,
   OUTOFBAND_1WIRE = 3,
   OUTOFBAND_BACKPLANE_0 = 4, //PXI_TRIG0 or AXIe TRIG0
   OUTOFBAND_BACKPLANE_1 = 5, //PXI_TRIG1 or AXIe TRIG1
   OUTOFBAND_BACKPLANE_2 = 6, //PXI_TRIG2 or AXIe TRIG2
   OUTOFBAND_BACKPLANE_3 = 7, //PXI_TRIG3 or AXIe TRIG3
   OUTOFBAND_BACKPLANE_4 = 8, //PXI_TRIG4 or AXIe TRIG4
   OUTOFBAND_BACKPLANE_5 = 9, //PXI_TRIG5 or AXIe TRIG5
   OUTOFBAND_BACKPLANE_6 = 10, //PXI_TRIG6 or AXIe TRIG6
   OUTOFBAND_BACKPLANE_7 = 11, //PXI_TRIG7 or AXIe TRIG7
   OUTOFBAND_BACKPLANE_8 = 12, //PXI_TRIG8 or AXIe TRIG8
   OUTOFBAND_BACKPLANE_9 = 13, //AXIe TRIG9
   OUTOFBAND_BACKPLANE_10 = 14, //AXIe TRIG10
   OUTOFBAND_BACKPLANE_11 = 15, //AXIe TRIG11
   OUTOFBAND_BACKPLANE_12 = 16, //AXIe TRIG12
};
```

Purpose

This enumeration defines the possible flow control sources for an Odi port. They are used in an array returned by `SS3_OdiPortGetCapabilities` to define supported flow control types.

enum LaneRate

```
enum LaneRate
{
   RATE_12R5G = 1, // 12.5 Gbps
   RATE_14R1G = 2 // 14.1 Gpbs
};
```

Purpose

This enumeration defines the possible lane rates of an Odi port. They are used in an array returned by `SS3_OdiPortGetCapabilities` to define supported lane rates on the port.

Chapter 8
.NET Interface

Introduction

The primary object used for .NET implementations is the cRecorder class. The class is in the Conduant.StreamStor3 namespace. This chapter provides member functions and constants for the cRecorder class. The class description includes a summary description of each member function followed by detailed detailed descriptions later in the chapter.

Class cRecorder

Namespace: `Conduant.StreamStor3`
Assembly: ssapi3.dll

Constructor(s):

```
cRecorder()
```

Constants:

```
static const int SS3_MAX_STRING;
static const int LOG_LEVEL_ERROR;
static const int LOG_LEVEL_TRACE;
static const int LOG_LEVEL_NONE;
```

Exceptions:

Member functions will throw exceptions if errors are encountered. The exceptions thrown are all based on the .NET Exception type. The following exception types are intentionally

thrown by the API if errors are encountered.

`ArgumentException` - *An argument to the function call is incorrect or not valid.*
`ArgumentOutOfRangeException` - *A specific argument (parameter) is not in the range of valid values. The parameter name is also available in the exception object.*
`InvalidOperationException` - *An unexpected operational error has occurred.*

Static Member Functions (synopsis):

```
static string GetApiVersion();
```
Return the API version in the form "1.0".

```
static string GetSdkVersion();
```
Return the SDK version in the form "1.0".

```
static uint FindDevices();
```
Return the number of detected StreamStor devices attached to the PCI Express fabric of the system.

```
static IList<SS3NetDevices> FindNetDevices();
```
Returns a list of StreamStor devices found on the local network.

```
static int SetLogLevel(int loglevel, uint32_t flags);
```
This function changes the logging level.of the API and / or Server

---

Member Functions (synopsis):

```
void ActivatePort(string portname);
void ActivatePort(string portname, uint serdesRate, ushort
maxBurst, ushort flowType);
```
Activate an external port so that it is enabled and sending or receiving data. Does not start recording.

```
int BindOutputPort(string recordingname, string recordedport,
```

```
string newport);
```
Change the port binding for an already recorded port. This allows the data recorded from one input port to be re-directed to a different port during playback. This binding will persist between sessions.

```
int ClearWriteProtect(string recordingname);
```
Clear the write protect from specified recording,

```
void Close();
```
Close the StreamStor device.

```
void CloseRecording();
```
Close an open recording. A recording must be open for reading or writing and must be closed before recording or playback.

```
void CreateRecording(string recordname, string portname)
void CreateRecording(string recordingname, string portname, ulong
filesize);
void CreateRecording(string recordingname,
Collections::Generic::List<cPortFile^> portlist);
```
Create a new recording. Multi port recordings must provide list of ports used with file length.

```
void DeactivatePort(string portname);
```
Deactivate an external port.

```
int DeleteRecording(string recordname);
```
Delete a recording.

```
int EraseRecording(string recordname, OWMODE overwrite);
```
Erase a recording with overwrite of data (not yet supported).

```
int EraseRecording(string recordname);
```
Erase a recording, no overwrite.

```
ulong GetBaseAddr();
```
Get the base address for PCI Express peer-to-peer transfer (not yet supported).

```
uint GetBaseRange();
```
Get the range (size) of the peer-to-peer memory window for a PCI Express port (not yet supported).

```
ulong GetDMABufferAddr();
```
Get the address of the DMA buffer.

```
uint GetDMABufferSize();
```
Get the size of the DMA buffer.

```
ulong GetDriveCapacity(int index);
```
Get the drive capacity (bytes) of the specified drive number.

```
int GetDriveCount();
```
Get the number of drives assigned to the open recorder.

```
string GetDriveModel(int drvIndex);
```
Get the model number of the specified drive.

```
string GetDriveSerial(int drvIndex);
```
Get the serial number of the specified drive.

```
string GetDriveVendor(int drvIndex);
```
Get the vendor name of the specified drive.

```
string GetDriverVersion();
```
Get the PCIe driver version number.

```
string GetFirmwareVersion();
```
Get the firmware version running on the StreamStor system.

```
string GetFirmwareDate();
```
Get the build date of the firmware software running on the StreamStor system.

```
string GetFPGAVersion();
```
Get the FPGA version for the StreamStor system.

```
ulong GetFreeCapacity();
```
Get the amount of free capacity available on the StreamStor system.

```
ulong GetLength(string recordingname, string portname);
```
Get the recorded data length of the specified recording / port. This version is intended for multi-port recordings.

```
ulong GetLength(string recordingname);
```
Get the recorded data length of the specified recording. This version is for single port recordings only.

```
string GetOSVersion();
```
This will return a string to identify the OS being used on the StreamStor system.

```
ulong GetPlayLength(string recordingname, string portname);
ulong GetPlayLength(string recordingname);
```
This will return the amount of data played back in real-time on the specified port or the total amount played on that port of the playback is completed. The playback length is volatile and only valid during or after a playback in the same session.

```
int GetPortCount();
```
Get the number of ports available in the system.

```
string GetPortName(int portindex);
```
This function will provide the port name string when provided an index from 0 to (GetPortCount() - 1). See "Ports" for an alternate iterable list of available ports.

```
int GetRecordedPortCount(string recordingname);
```
This function will return the number of ports defined in this recording for input. These ports may or may not have recorded data.

```
string GetRecordedPortName(string recordingname, int indx);
```
This function will return the name of the indicated port with recorded data.

```
string GetRecorderName();
```
This function will provide the system name. A default name is assigned at the factory that can be changed with SetRecorderName.

```
string GetRecorderType();
```
This function returns a string identifying the StreamStor system type implemented.

```
int GetRecordingCount();
```
This function returns the number of recordings on the recorder. See "Recordings" for an alternate iterable list of recordings / ports.

```
string GetRecordingName(int recordindex);
```
This function will provide the recording name when provided an index from 0 to (GetRecordingCount() - 1). See "Recordings" for an iterable list of recordings / ports.

```
time_t GetRecordingTimestamp(string recordname);
```
Get the timestamp of when the specified recording was last modified.

```
ulong GetProgress();
```
This function will return the amount of data that has been transferred in an in progress record or playback operation.

```
ulong GetProgress(string portname);
```
This function will return the amount of data that has been transferred in an in progress record or playback operation.

```
uint GetSerialNumber();
```
Get the serial number of the StreamStor system.

```
uint GetSystemErrorCode();
```
Get the system error code. Value only valid when the SystemError flag is set (see IsSystemErrorSet()).

```
time_t GetTime();
```
Get the system time on the StreamStor recorder. This is seconds since 00:00:00 Jan 1, 1970 UTC.

```
ulong GetTotalCapacity();
```
Get the total capacity of the storage on the StreamStor system.

```
void* GetWindowAddr();
```
Get the user mapped address of the peer-to-peer memory window of the StreamStor system on the PCI Express bus (not yet implemented).

```
bool IsBufferOverflowed();
```
This will return true if the buffer has overflowed during a record operation.

```
bool IsStorageOverflowed();
```
This will return true if the storage space filled to capacity during a record operation.

```
bool IsInputPortBound();
```
This will return true if at least one port has been bound for input (recording).

```
bool IsOpen();
```
This will return true if there is a StreamStor device open.

```
bool IsReady();
```
This will return true if the StreamStor system is opened and ready for operation.

```
bool IsRecording();
```
This will return true if the StreamStor system is in record mode and actively recording.

```
bool IsRecordingOpen();
```
This will return true if a recording is currently open for reading or writing.

```
bool IsRecordingOpenForRead();
```
This will return true if a recording is currently open for reading.

```
bool IsRecordingOpenForWrite();
```
This will return true if a recording is currently open for writing.

```
bool IsPlaying();
```
This will return true if the StreamStor system is actively playing back a recording.

```
bool IsSystemErrorSet();
```
This will return true if a system error has occurred.

```
bool IsUnitAttentionSet();
```
This will return true if unit attention has been asserted as the result of a power loss or restart of the recording system.

```
bool IsWriteProtected(string recordingname);
```
This will return true if the specified recording has been protected by setting write protect.

```
int Open(const char *address, ushort port);
```
This function will open a StreamStor recorder at the specified IP address (or hostname) and port number.

```
int Open(string address, ushort port);
```
This function will open a StreamStor recorder at the specified IP address (or hostname) and port number.

```
int Open();
```
This function will open a StreamStor recorder on the PCI Express interface. This form assumes only one StreamStor is present on the PCI Express interface.

```
int Open(int devindex);
```
This function will open a StreamStor recorder on the PCI Express interface. The devindex is a one (1) based index if more than one StreamStor device is present.

```
int OpenRecording(string name, bool forread);
```
This function will open a recording for read or write. For reading, "forread" should be set true, or false for writing. This form assumes there is only a single port recorded.

```
int OpenRecording(string recordingname);
```
This function will open a recording for read or write. This form assumes there is only a single port recorded and the recording is being opened for reading.

```
int OpenRecording(string name, string portname, bool forread);
```
This function will open a recording for read or write. For reading, "forread" should be set true, or false for writing.

```
int OpenRecording(string recordingname, string portname);
```
This function will open a recording for read or write. This form assumes the recording is being opened for reading.

```
int Playback(string recordingname);
```
This function will start playback of the specified recording from the beginning of the data.

```
int Playloop(string recordingname);
```
This function will start playback of the specified recording and loop continuously until stopped.

```
int Playloop(string recordingname, uint loops);
```
This function will start playback of the specified recording and loopback so that the data is played back completely the number of times specified by the "loops" parameter.

```
uint Read(array<byte>^ BuffAddr, ulong Addr, uint Length);
```
This function will read data from the open recording and copy the data to the specified buffer.

```
int Record(string recordingname);
```
This function will start recording from the configured port(s) until stopped or storage space is exhausted.

```
int Record(string Recordingname, bool WrapEnable);[1]
```
This function will start recording from the configured port(s). When true, WrapEnable configures the recorder to write continuously even when space is exhausted by overwriting old data (not yet supported).

```
int RenameRecording(string name, string newname);
```
This function changes the name of an existing recording.

```
int Reset();
```
This function will reset the StreamStor system. This will also close the recorder and require a new Open request.

```
int SetPlayContext(string recordingname, ulong offset, ulong
length);
```
This function sets the playback parameters for a recording with a single recorded port.

```
int SetPlayContext(string recordingname, string portname, ulong
offset, ulong length);
```
This function sets the playback parameters for a port in a recording.

```
int SetRecorderName(string name);
```
This function changes the name of the recorder.

```
int SetTime(time_t newtime);
```
This function will update the system time on the StreamStor system. The time should be provided as seconds since 00:00:00 Jan 1, 1970 UTC.

```
int SetTime();
```
This function will set the time on the StreamStor system using the time from the local system.

```
int SetWriteProtect(string recordingname);
```
This function will set write protection on the specified recording. When write protect is set, the recording cannot be opened for writing or erased.

```
int Stop();
```
This function is used to stop a recording or playback operation. It should always be called to end a playback or recording even when the operation has automatically halted.

```
void TruncateRecording(string recordingname)
```
This function is used to reduce the file size of a recording.

```
int Write(void *BufferAddr, uint XferLength);
```
This function is used to write data from a user buffer to a recording data set that has been
previously opened for writing.

Collections / Lists:

```
Ports
```
Enumerable list of ports available (string).

```
Recordings
```
Enumerable list of recordings, member "Name" holds recording name. Member "Ports"

is an enumerable list of recorded port names (String).

`Recordings::Count`

Retrieve the number of recordings on the recorders

`RecordingsByName`

Enumerable dictionary of recordings, indexed by name. Member "Ports" is an enumerable list of recorded port names (string).

Member Functions (details):

cRecorder::ActivatePort

Syntax:

```
void ActivatePort(string portname);
void ActivatePort(string portname, uint serdesRate, ushort
maxBurst, ushort flowType);
```

Description:

This function activates an external port and enables it for sending/receiving. Note that this function alone does not start recording from a port. The port must be included in a recording when the recording is created in order to be used for recording. A port must be activated before a recording is started.

Parameters:

`portname` - name of an external port to be activated.
`serdesRate` - the serial bit rate to program on this port. See hardware documentation for bit rate support. Bit rates are expressed as an unsigned integer in mbps (i.e. 12500 is 12.5 Gbps).
`maxburst` - this value sets a maximum burst size in bytes on the port. See hardware documentation for burst size support.
`flowType` - this value sets the type of flow control to for ports that support flow control. See hardware documentation for flow control type supported.

Return Value:

none

See Also:

`DeactivatePort`

Usage:

cRecorder::BindOutputPort

Syntax:

```
void BindOutputPort(string recordingname, string portname, string
newportname)
```

Description:

This function changes the binding of a recorded port in an existing recording so that subsequent playback will use the new port rather than the current assigned port. The default output port is the same port used when recording as set by CreateRecording. The list of ports available are unique to each system and can be listed by enumerating the "Ports" list or by using GetPortCount and GetPortName.

Parameters:

`recordingname` - name of recording that will be changed.
`portname` - name of the currently recorded port to be changed.
`newportname` - name of port to be reassigned as the new output port.

Return Value:

none

See Also:

`Playback`, `GetPortCount`, `GetPortName`, `Ports`

Usage:

cRecorder::ClearWriteProtect

Syntax:

```
void ClearWriteProtect(string recordingname)
```

Description:

This function clears the write protection from a recording. Write protection can be set using

the SetWriteProtect function and prevents a recording from being erased or deleted by other commands in the API. The status of write protection can be checked using the `IsWriteProtected` function.

Parameters:

recordingname - name of the recording to have write protection cleared.

Return Value:

none

See Also:

`SetWriteProtect`, `IsWriteProtected`

Usage:

cRecorder::Close

Syntax:

```
void Close()
```

Description:

This function closes the connection to the StreamStor system. This function should be called whenever the user application is done using the StreamStor system.

Parameters:

none

Return Value:

none

See Also:

`Open`, `NetOpen`

Usage:

cRecorder::CloseRecording

Syntax:

```
void CloseRecording()
```

Description:

This function closes a recording that has been previously opened for reading or writing with OpenRecording. The status of whether a recording is currently open can be tested using the `IsRecordingOpen`, `IsRecordingOpenForRead` and `IsRecordingOpenForWrite` functions. The system cannot record or playback while a recording is open.

Parameters:

none

Return Value:

none

See Also:

`OpenRecording`, `IsRecordingOpen`, `IsRecordingOpenForRead`, `IsRecordingOpenForWrite`

Usage:

cRecorder::CreateRecording

Syntax:

```
void CreateRecording(string recordingname)
void CreateRecording(string recordingname, string portname)
void CreateRecording(string recordingname, List<cPortFile>
portlist)
```

Description:

This function creates a new, empty recording on the StreamStor system. The provided name must be unique from any other recording names. The first form assumes a system supporting only a single port. The second form accepts a port name to define the port that will be recorded. The final form accepts a list of port names that will be recorded. The `cPortFile` class includes a length parameter for each port to define the number of bytes to allocate for each.

Parameters:

`recordingname` - name of the new recording.
`portname` - port name to use when recording to this recording.
`List<cPortFile>` - List of `cPortFile` classes defining the ports and lengths to use in the new recording.

Return Value:

none

Related Class

```
cPortFile
   Members:
      string PortName;
      ulong  FileBytes;

   Constructors:
      cPortFile()
      cPortFile(const cPortFile c)
```

See Also:

Usage:

---

cRecorder::DeactivatePort

Syntax:

```
void DeactivatePort(string portname)
```

Description:

This function deactivates an external port. The port must be activated again prior to attempting to record with this port.

Parameters:

`portname` - name of the port to be deactivated.

Return Value:

none

See Also:

`ActivatePort`

Usage:

cRecorder::DeleteRecording

Syntax:

```
void DeleteRecording(string recordingname)
```

Description:

This function deletes a recording from the StreamStor system. If the recording is the last remaining recording it will be erased and renamed to a default name so that the system always has at least one recording.

Parameters:

`recordingname` - name of the recording to be deleted.

Return Value:

none

See Also:

`EraseRecording`

Usage:

cRecorder::FindNetDevices

Syntax:

```
IList<SS3NetDevices> FindNetDevices()
```

Description:

This function searches the local network for any StreamStor devices accessible on the network. Each network interface that is connected is searched according to the active network address mask active for that interface. The function returns a list of SS3NetDevice classes.

Parameters:

Return Value:

```
IList<SS3NetDevice>
```

See Also:

Usage:

```
Return class (SS3NetDevice)
public class SS3NetDevices
{
   public string SystemName;
   public string IPv4Address;
   public string IPv6Address;
   public ushort Port;
}
```

cRecorder::GetDMABufferAddr

Syntax:

```
ulong GetDMABufferAddr(uint sizeBytes)
```

Description:

This function returns the address of the DMA buffer provided for reading and writing over the PCI Express interface.

Parameters:

`sizeBytes` - Size of the requested DMA buffer in bytes.

Return Value:

DMA mapped local address.

See Also:

`GetDMABufferSize`

Usage:

cRecorder::GetDMABufferSize

Syntax:

```
int GetDMABufferSize(ulong address)
```

Description:

This function returns the size in bytes of the DMA buffer allocated with `GetDMABufferAddr`. The DMA buffer may be used when reading or writing data to a recording.

Parameters:

Return Value:

Size of buffer in bytes.

See Also:

`GetDMABufferAddr`

Usage:

cRecorder::GetDriveCapacity

Syntax:

```
ulong GetDriveCapacity(int index)
```

Description:

This function returns the drive capacity of the drive indicated by the supplied index. Indexing starts at zero and must be less than the drive count as returned by `GetDriveCount`.

Parameters:

`index` - index number of the drive to get capacity.

Return Value:

none

See Also:

`GetDriveCount`

Usage:

cRecorder::GetDriveCount

Syntax:

```
int GetDriveCount()
```

Description:

This function returns the number of drives being used by the StreamStor system.

Parameters:

none

Return Value:

Number of drives

See Also:

Usage:

cRecorder::GetDriveModel

Syntax:

```
string GetDriveModel(int index)
```

Description:

This function returns the model number of a drive on the StreamStor system. The index numbers start at zero and must be less than the count returned by GetDriveCount.

Parameters:

`index` - index number of the drive

Return Value:

String representing the drive model number.

See Also:

`GetDriveCount`, `GetDriveSerial`, `GetDriveVendor`

Usage:

cRecorder::GetDriveSerial

Syntax:

```
string GetDriveSerial(int index)
```

Description:

This function returns the serial number of a drive on the StreamStor system. The index numbers start at zero and must be less than the count returned by GetDriveCount.

Parameters:

`index` - index number of the drive

Return Value:

String representing the drive serial number

See Also:

`GetDriveCount`, `GetDriveModel`, `GetDriveVendor`

Usage:

cRecorder::GetDriveVendor

Syntax:

```
string GetDriveVendor(int index)
```

Description:

This function returns the vendor name (manufacturer) of a drive on the StreamStor system. The index numbers start at zero and must be less than the count returned by `GetDriveCount`.

Parameters:

`index` - index number of the drive

Return Value:

String representing the drive vendor.

See Also:

`GetDriveCount`, `GetDriveSerial`, `GetDriveModel`

Usage:

cRecorder::GetDriverVersion

Syntax:

```
string GetDriverVersion()
```

Description:

This function returns the version number of the driver software being used for the PCI Express interface to the system.

Parameters:

none

Return Value:

String representing the driver version.

See Also:

Usage:

cRecorder::GetFirmwareVersion

Syntax:

```
string GetFirmwareVersion()
```

Description:

This function returns the version number of the firmware running on the StreamStor system.

Parameters:

none

Return Value:

String representing the firmware version.

See Also:

Usage:

cRecorder::GetFreeCapacity

Syntax:

```
ulong GetFreeCapacity()
```

Description:

This function returns the free capacity available on the StreamStor system. The amount returned does not account for overhead losses for new recordings due to file system records and other internal data and should be relied upon to provide an exact amount of space available for new recordings.

Parameters:

none

Return Value:

Free space available in bytes.

See Also:

Usage:

cRecorder::GetFPGAVersion

Syntax:

```
string GetFPGAVersion()
```

Description:

This function returns the version number of the FPGA used in the StreamStor system.

Parameters:

none

Return Value:

String representing the FPGA version.

See Also:

Usage:

cRecorder::GetLength

Syntax:

```
ulong GetLength(string recordingname)
ulong GetLength(string recordingname, string portname)
```

Description:

This function returns the recorded data length of the indicated recording. If a recording has multiple ports recorded the portname of interest should be provided. The length is returned as the number of bytes.

Parameters:

`recordingname` - name of an existing recording to query.

`portname` - name of the recorded port in the recording to query.

Return Value:

Length of the recorded data in bytes.

See Also:

Usage:

cRecorder::GetPlayLength

Syntax:

```
ulong GetPlayLength(string recordingname)
ulong GetPlayLength(string recordingname, string portname)
```

Description:

This function will return the amount of data played back on the specified port in the most recent playback operation. The port name does not need to be supplied for single port recordings. The play length is volatile and only valid if a playback has occurred since the last reset or power event.

Parameters:

`recordingname` - name of an existing recording to query.
`portname` - name of the recorded port in the recording to query.

Return Value:

Amount of data played in bytes.

See Also:

Usage:

cRecorder::GetPortCount

Syntax:

```
int GetPortCount()
```

Description:

This function returns the number of available ports on the SteamStor system. These ports can be specified for use in recording data using `CreateRecording`.

Parameters:

none

Return Value:

Number of ports available.

See Also:

Usage:

cRecorder::GetPortName

Syntax:

```
string GetPortName(int index)
```

Description:

This function returns the name of a recording / playback port on the SteamStor system. These ports can be specified for use in recording data using `CreateRecording`. The index starts at zero (0) and must be less than the number of available ports returned by `GetPortCount`.

Parameters:

`index` - index number of the port to get name.

Return Value:

String with name of the port.

See Also:

Usage:

cRecorder::GetPortStatus

Syntax:

```
PortStatus GetPortStatus(string portname)
```

Description:

This function returns the status of an external port.

Parameters:

`portname` - name of the port to check status.

Return Value:

Bitmap of port status (see enum PortStatus)

See Also:

Usage:

cRecorder::GetProgress

Syntax:

```
ulong GetProgress()
ulong GetProgress(string portname)
ulong GetProgress(string portname, int recorderindex)
```

Description:

This function returns the amount of data transferred in an ongoing recording or playback operation. When only a single port is being used the `portname` is not required. The value returned is an approximation of the transferred amount since the value will be constantly changing. When using a non-chained system with more than one recorder, the recorder index allows this function to retrieve progress from any active recorder with having to use `cRecorder::SelectedRecorder(recorderindex)` first.

Parameters:

`portname` - name of port to get progress length.
`recorderindex` – index of recorder.

Return Value:

Length in bytes.

See Also:

`SelectedRecorder, Record, Playback`

Usage:

cRecorder::GetRecordedPortCount

Syntax:

```
int GetRecordedPortCount(string recordingname)
```

Description:

This function returns the number of ports with recorded data in the named recording. This value is also available in the Count property of the enumerable list: `Recordings[recordingname].Ports.Count.`

Parameters:

`recordingname` - name of recording to get number of recorded ports.

Return Value:

Number of recorded ports.

See Also:

Usage:

cRecorder::GetRecordedPortName

Syntax:

```
string GetRecordedPortName(string recordingname, int index)
```

Description:

This function returns the name of the recorded port in a recorded port by index. The index starts at zero(0) and must be less than the count returned by `GetRecordedPortCount`. The recorded ports are also available in an enumerable list `Recordings[].Ports`.

Parameters:

`recordingname` - name of recording.
`index` - index of recorded port to get name.

Return Value:

String with port name.

See Also:

Usage:

cRecorder::GetRecorderCount

Syntax:

```
uint GetRecorderCount()
```

Description:

This function returns the number of recorders in the system. When a system is chained, this is the number of recorders used in the chain. When not chained, this is the number of independent recorders in the attached StreamStor system. The `SelectedRecorder` function can be used to set the currently selected recorder or to query which recorder is currently selected.

Parameters:

none

Return Value:

Number of recorders in this system.

See Also:

`cRecorder::SelectedRecorder`

Usage:

cRecorder::GetRecorderName

Syntax:

```
string GetRecorderName()
```

Description:

This function returns the name of the StreamStor recorder. A custom name can be assigned using `SetRecorderName`. If no custom name has been assigned a default name is used for the recorder.

Parameters:

none

Return Value:

String with recorder name.

See Also:

`SetRecorderName`

Usage:

cRecorder::GetRecorderType

Syntax:

```
string GetRecorderType()
```

Description:

This function returns a description of the specific type of StreamStor recorder in use. The type of recorder will vary depending on the type of ports and other customizations implemented.

Parameters:

none

Return Value:

String with recorder type description.

See Also:

Usage:

cRecorder::GetRecordingCount

Syntax:

```
int GetRecordingCount()
```

Description:

This function returns the number of recordings on the StreamStor system. the recordings are also available in an enumerable list "`Recordings`".

Parameters:

none

Return Value:

Number of recordings.

See Also:

Usage:

cRecorder::GetRecordingName

Syntax:

```
string GetRecordingName(int index)
```

Description:

This function returns the name of a recording by index. The index starts at zero(0) and must be less than the count returned by `GetRecordingCount`. The recording names are also available in an enumerable list "`Recordings[].Name`".

Parameters:

`index` - index of recording to get name.

Return Value:

String with recording name.

See Also:

Usage:

cRecorder::GetRecordingTimestamp

Syntax:

```
time_t GetRecordingTimestamp(string recordingname)
```

Description:

This function returns the time that was logged when the recording was started. The time is returned as the number of seconds since January 1, 1970 UTC.

Parameters:

`recordingname` - name of recording.

Return Value:

Time in seconds since Jan, 1, 1970 UTC

See Also:

Usage:

cRecorder::GetSerialNumber

Syntax:

```
uint GetSerialNumber()
```

Description:

This function returns the serial number of the StreamStor Cobra recorder board.

Parameters:

none

Return Value:

Serial number as integer.

See Also:

Usage:

cRecorder::GetTotalCapacity

Syntax:

```
ulong GetTotalCapacity()
```

Description:

This function returns the total capacity of all storage space available on the StreamStor recorder.

Parameters:

none

Return Value:

Total capacity in bytes.

See Also:

Usage:

cRecorder::IsBufferOverflowed

Syntax:

```
bool IsBufferOverflowed();
```

Description:

If this function returns true, the system has run into a buffer overflow condition. A buffer overflow generally occurs when the incoming data rate is higher than the rate of writing to storage. A properly defined system should never see this occur in normal operation.

Parameters:

none.

Return Value:

True if a buffer overflow has occurred, False otherwise.

Usage:

cRecorder::IsPlaying

Syntax:

```
bool IsPlaying();
```

Description:

This function can be used to indicate if the system is currently in playback of a recording. If a playback ends due to the programmed playback size being reached, this function will return false to indicate that the playback has ended.

Parameters:

none.

Return Value:

True if playback in progress, False otherwise.

Usage:

cRecorder::IsRecording

Syntax:

```
bool IsRecording();
```

Description:

If this function returns true, the system is currently recording data. If programmed to wrap, the record operation will never exit and this function will always return true.

Parameters:

none.

Return Value:

True if recording is in progress, False otherwise.

Usage:

cRecorder::IsRecordingOpen

Syntax:

```
bool IsRecordingOpen();
```

Description:

If this function returns true, there is a recording currently open for reading or writing. Any open recording must be closed before a playback or record operation can be started.

Parameters:

none.

Return Value:

True if a recording is open, False otherwise.

Usage:

cRecorder::IsRecordingOpenForRead

Syntax:

```
bool IsRecordingOpenForRead();
```

Description:

If this function returns true, there is a recording currently open for reading. Any open recording must be closed before a playback or record operation can be started.

Parameters:

none.

Return Value:

True if a recording is open for read, False otherwise.

Usage:

cRecorder::IsRecordingOpenForWrite

Syntax:

```
bool IsRecordingOpenForWrite();
```

Description:

If this function returns true, there is a recording currently open for writing. Any open recording must be closed before a playback or record operation can be started.

Parameters:

none.

Return Value:

True if a recording is open for write, False otherwise.

Usage:

cRecorder::IsStorageOverflowed

Syntax:

```
bool IsStorageOverflowed();
```

Description:

If this function returns true, the current or last recording reached the storage capacity limits. A recording started with wrap enabled will never reach this state.

Parameters:

none.

Return Value:

True if a recording has used all available storage, False otherwise.

Usage:

cRecorder::IsSystemErrorSet

Syntax:

```
bool IsSystemErrorSet();
```

Description:

If this function returns true, a critical system error has occurred. System errors indicate an unexpected hardware, storage, or software problem has occurred.

Parameters:

none.

Return Value:

True if a system error has occurred, False otherwise.

Usage:

cRecorder::IsUnitAttentionSet

Syntax:

```
bool IsUnitAttentionSet();
```

Description:

If this function returns true, a unit attention event has occurred on the system. This is an indication that a power cycle or reset has occurred on the StreamStor system since the session was started.

Parameters:

none.

Return Value:

True if unit attention is active, False otherwise.

Usage:

cRecorder::IsValidRecordingName

Syntax:

```
bool IsValidRecordingName(string checkname);
```

Description:

If this function returns true, the provided name is a valid recording on the system. If the function returns false, the name is not valid. Provides a way to validate a recording name before using in other functions.

Parameters:

`checkname` - Recording name to check.

Return Value:

True if name valid, False otherwise.

Usage:

cRecorder::Open

Syntax:

```
void Open(string ip_addr, ushort ip_port);
void Open(int dev_index);
void Open(int dev_index, int adapter);
```

Description:

This function is used to open a specific StreamStor system. the system can be opened over a network connection by supplying an IP address and port number or it can be opened on the PCI Express interface using a device index (when more than a single recorder is connected to one adapter) and optionally an adapter number (0 base index). When an adapter number is not supplied it is defaulted to adapter 0. The StreamStor system must be opened before any other operations can be performed. When multiple recorders are resident in a common chassis, the adapter number will be common and the device index will increment for each recorder. When using the network connection, the default port number is incremented for each recorder under the same IP
address.

Parameters:

`ip_addr` - String with IP address or hostname of StreamStor system.
`ip_port` - port number that the system is configured to listen (default is 59427).
`dev_index` - index of StreamStor system on PCI Express (starts at 1).
`adapter` - index of host system Dolphin SISCI PCI Express adapter (starts at 0).

Return Value:

none

Usage:

cRecorder::OpenRecording

Syntax:

```
void OpenRecording(string recordingname, bool forRead)
void OpenRecording(string recordingname, string portname, bool
forRead)
```

Description:

This function opens a recording data set for reading or writing data. Single port recordings may omit the `portname` parameter. A recording must be opened before data be written or read from it. The recording must also be closed using `CloseRecording` before `Record`, `Playback` and other operations can proceed.

Parameters:

`recordingname` - Name of recording to open.
`portname` - Name of recorded port to open.
`forRead` - determines if recording is being opened for read (true) or write (false).

Return Value:

none

See Also:

Usage:

cRecorder::Playback

Syntax:

```
void Playback(string recordingname)
```

Description:

This function starts a playback of a recording. The recorded data will playback to the port associated with the data when recorded or when re-assigned using the `BindOutputPort` function. Playback continues until all data has been played or `Stop` is called. The starting offset and length can be changed using the `SetPlayContext` function.

Parameters:

`recordingname` - Name of recording to playback.

Return Value:

none

See Also:

Usage:

cRecorder::Read

Syntax:

```
uint Read(array<byte>^ buffer, ulong offset, uint length)
```

Description:

This function requires a previously opened recording (for read). The read function will move data from the open recording to a user defined buffer. The offset defines the starting point in the recorded data and the length defines the requested amount of data to read. The actual amount read will be returned.

Parameters:

`buffer` - user allocated buffer to hold the data.
`offset` - offset location in the recorded data to start reading
`length` - requested amount to read.

Return Value:

actual amount of data read (bytes)

See Also:

`OpenRecording`

Usage:

cRecorder::Record

Syntax:

```
void Record(string recordingname, bool wrapenable)
void Record(string recordingname)
```

Description:

This function starts a recording on the StreamStor system. If multiple ports have been assigned for input than a multi-port recording will be created. The recording will continue until storage space is exhausted or `Stop` is called.

Parameters:

`recordingname` - name of recording to use for this recording (must be empty).
`wrapenable` - true indicates recording should overwrite old data when storage limit reached (not yet supported).

Return Value:

none

See Also:

CreateRecording

Usage:

cRecorder::RenameRecording

Syntax:

```
void Record(string currentname, string newname)
```

Description:

This function can be used to rename an existing recording on the StreamStor system.

Parameters:

`currentname` - name of an existing recording.
`newname` - new name for this recording.

Return Value:

none

See Also:

Usage:

cRecorder::SelectedRecorder

Syntax:

```
int SelectedRecorder()
void SelectedRecorder(int recorder)
```

Description:

This function either returns the index of the currently selected recorder or is used to select a specific recorder (when more than one is present). The index numbers start at 0 so that the maximum index is the value returned from `cRecorder::GetRecorderCount()` - 1.

Parameters:

`recorder` – index of recorder to select.

Return Value:

`index` – index of recorder currently selected.

See Also:

Usage:

cRecorder::SetRecorderName

Syntax:

```
void SetRecorderName(string recordername)
```

Description:

This function assigns a user defined name to the StreamStor system. If no name is assigned a default name is used. The name can be retrieved using the GetRecorderName function.

Parameters:

`recordername` - name to use for recorder.

Return Value:

none

See Also:

Usage:

cRecorder::SetPlayContext

Syntax:

```
void SetPlayContext(string recordingname, string portname, ulong
offset, ulong length)
void SetPlayContext(string recordingname, ulong offset, ulong
length)
```

Description:

This function can be used to define a playback offset and/or length to allow playback of recording subsets. The play context is saved on a persession basis only and is not preserveed between sessions. If the recording has multiple recorded ports, a context can be set for each port independently. If a play context is established for a `Playloop` operation, the playback will always loop back to the offset specified (not to offset 0).

Parameters:

`recordingname` - name of recording to assign this context.
`portname` - name of port to assign this context.
`offset` - offset into the recorded data to start playback.
`length` - length of data to play (0 for play to end)

Return Value:

none

See Also:

Usage:

cRecorder::SetTime

Syntax:

```
void SetTime(time_t seconds)
void SetTime()
```

Description:

This function is used to set the time on the StreamStor system. The time is used to timestamp recordings. If no time is provided, the function will use the time on the user computer to set the time on the StreamStor system. The time is in seconds since January 1, 1970 UTC.

Parameters:

`seconds` - time to set in seconds since January 1, 1970 UTC.

Return Value:

none

See Also:

Usage:

cRecorder::SetWriteProtect

Syntax:

```
void SetWriteProtect(string recordingname)
```

Description:

This function sets write protection on the indicated recording. This protect the recording against erase or delete. this protection only works from within the StreamStor API and does not protect the data if the files or disk are accessed by some other method.

Parameters:

`recordingname` - name of recording to protect.

Return Value:

none

See Also:

Usage:

cRecorder::Stop

Syntax:

```
void Stop()
```

Description:

This function can be used to stop an in progress record or playback operation. It should not be used when reading or writing to/from an open recording.

Parameters:

none

Return Value:

none

See Also:

Usage:

cRecorder::TruncateRecording

Syntax:

```
void TruncateRecording(string recordingname)
```

Description:

This function is used to reduce a recordings file size down to only the amount needed to hold the currently recorded data.

Parameters:

`recordingname` - recording name to truncate.

Return Value:

none

See Also:

Usage:

cRecorder::Write

Syntax:

```
uint Write(array<byte>^ buffer, uint length)
```

Description:

This function is used to write data to a recording that has been opened for writing. Applications should always check the returned length when this function completes, the return value will indicate how much data was actually written in the case where a recording has exceeded capacity.

Parameters:

`buffer` - user allocated buffer holding the data to write to the recording.
`length` - length of data to write.

Return Value:

Number of bytes written

See Also:

Usage:

Class cOdiRecorder

Namespace: `Conduant.StreamStor3`
Assembly: ssapi3.dll

Overview

The cOdiRecorder class is a higher level class that manages the lower level cRecorder class to simplify operations for ODI recording applications. The class hides many of the complexities of managing packets and the capture of packet size information. The ODI recorder uses the "ODI1" port for all input (record) operations and "ODI2" for all playback operations.

Constructors

```
cOdiRecorder(int pcie_index)
cOdiRecorder(string ip_addr, ushort ip_port)
```

Collections (iList interface)

Recordings	List of recordings
string name	Recording name
Recordings[*index*].Ports	List of recorded ports in

	recording
string name	Port name

Member Functions (synopsis)

```
void CloseRecording();
```
Close a previously opened recording.

```
void CreateRecording(string recordingname );
```
Create a new recording using all available capacity.

```
void CreateRecording(string recordingname, ulong size);
```
Create a new recording with size (bytes) as requested.

```
void DeleteRecording(string recordingname);
```
Delete a recording.

```
void EraseRecording(string recordingname);
```
Erase the data in a recording.

```
ulong GetDriveCapacity(int index);
```
Get the capacity of a disk in the reorder.

```
string GetAPIVersion();
```
Get the API version string.

```
int GetDriveCount();
```
Get the number of disks in the recorder.

```
string GetDriveModel(int index);
```
Get the drive model name.

```
string GetDriveSerial(int index);
```
Get the drive serial number.

```
string GetDriveVendor(int index);
```
Get the drive manufacturer name.

```
string GetFirmwareVersion();
```
Get the recorder firmware version number.

```
string GetFPGAVersion();
```
Get the recorder FPGA version number.

```
ulong GetFreeCapacity();
```
Get the free (unused) capacity remaining on the recorder storage.

```
ulong GetLength(string recordingname);
```
Get the total size (bytes) of the recorded packets.

```
ulong GetPacketCount(string recordingname);
```
Get the number of packets in the recording.

```
uint GetPacketErrorCount();
```
Return the number of packets that were detected as defective in the latest recording or playback.

```
ulong GetPlayLength(string recordingname);
```
Get the length of the last playback (bytes).

```
string GetRecorderName();
```
Get the user defined recorder name.

```
int GetRecorderCount();
```
Get the number of recorders in the system.

```
int GetRecordingCount();
```
Get the number of recordings stored on the system.

```
string GetRecordingName(int index);
```
Get the recording name.

```
ulong GetProgress();
```
Get the in-progress recording or playback length (estimate).

```
ulong GetProgress(int recorderindex);
```
Get the in-progress recording or playback length (estimate) on the specified recorder.

```
uint GetSystemErrorCode();
```
Get the system error code, only valid if system error status active.

```
ulong GetTotalCapacity();
```
Get the total storage capacity (bytes) available on the recorder.

```
bool IsBufferOverflowed();
```
Check for recording that has overflowed DRAM buffer capacity limits.

```
bool IsOpen();
```
Check to see if system has been opened for access.

```
bool IsPlaying();
```
Check to see if system is actively playing back data.

```
bool IsReady();
```
Check to see if system is ready for operation.

```
bool IsRecording();
```
Check to see if system is actively recording data.

```
bool IsRecordingOpen();
```
Check to see if a recording is currently open for reading or writing.

```
bool IsRecordingOpenForRead();
```
Check to see if a recording is currently open for reading.

```
bool IsRecordingOpenForWrite();
```
Check to see if a recording is currently open for writing.

```
bool IsStorageOverflowed();
```
Check for recording that has reached capacity limits.

```
bool IsSystemErrorSet();
```
Check to see if a system error has been reported.

```
bool IsUnitAttentionSet();
```
Check to see if the system has been reset since the session began.

```
void OpenRecording(string recordingname, bool forRead);
```
Open a recording for reading or writing (ReadPacket / WritePacket) .

```
void Playback(string recordingname);
```
Begin playback of a recording.

```
void Playloop(string recordingname);
```
Begin continuous playback of a recording. Use Stop to end playback.

```
void Playloop(string recordingname, uint loops);
```
Begin continuous playback of a recording the number of times specified

```
int ReadPacket(ulong pindex, void* buffer, uint bufsize);
```
Read a packet from the open recording into the buffer provided. Returns size of packet read.

```
void Reconfigure(string configname);
```
Reconfigure the recorder using the named configuration file.

```
void Record(string recordingname);
```
Start recording data from the ODI port to the recording specified.

```
void RenameRecording(string existing, string newname);
```
Rename an existing recording to a new name.

```
int SelectedRecorder();
```
Return the currently selected recorder.

```
void SelectedRecorder(int recorderindex);
```
Set the currently selected recorder.

```
void SetRecorderName(string newname);
```
Set a user defined name for the recorder.

```
void Stop();
```
Stop a record or playback operation.

```
int WritePacket(void* buffer, uint size);
```
Write a packet to the currently open recording.

Member Functions (detailed)

cOdiRecorder (constructors)

Syntax:

```
cOdiRecorder(string address, ushort port)
cOdiRecorder(int pci_index)
```

Description:

These constructors are used to create the recorder object and open a connection the the desired recording system. The first form of the constructor is used to connect to a recording system over the network by providing an IP address (or hostname) and the port number. The default port number for StreamStor systems is 59427. If multiple StreamStor recorders are installed in a single system, the port number is incremented for each successive recorder. The second form of the constructor is used to connect to a StreamStor system across a cabled PCI Express connection.

Parameters:

`address` - The IP address of hostname of the system to connect.
`port` - The IP port to use to connect to the StreamStor system.
`index` - The index number of the recorder to open (starts at 1)

Return Value:

none

Usage:

cOdiRecorder::CloseRecording

Syntax:

```
void CloseRecording();
```

Description:

This function is used to close a previously opened recording. A recording must be opened to allow reading or writing of packets. The recording must be closed before any record or playback operation.

Parameters:

Return Value:

none

Usage:

cOdiRecorder::CreateRecording

Syntax:

```
void CreateRecording(string recordingname);
void CreateRecording(string recordingname, ulong size);
```

Description:

This function is used to create a new recording on the StreamStor system. The provided name must be unique from any other recording names already on the system. if no size is provided, the system will create the largest possible recording with remaining space.

Parameters:

`recordingname` - name to assign to the new recording.

`size` - size of the recording to create (bytes)

Return Value:

none

Usage:

cOdiRecorder::DeleteRecording

Syntax:

```
void DeleteRecording(string recordingname);
```

Description:

This function is used to delete an existing recording on the StreamStor system. If the recording is the only recording currently on the system it will be erased and renamed to a default name.

Parameters:

`recordingname` - name of recording to delete from the system.

Return Value:

none

Usage:

cOdiRecorder::EraseRecording

Syntax:

```
void EraseRecording(string recordingname);
```

Description:

This function is used to erase a recording on the StreamStor system. Any recorded data is erased but the recording itself will not be removed. A recording must be erased (if not already empty) to allow a new record.

Parameters:

`recordingname` - name to assign to the new recording.

Return Value:

none

Usage:

cOdiRecorder::GetAPIVersion

Syntax:

```
string GetAPIVersion();
```

Description:

This function is used to retrieve the version number of the current API (ssapi3.dll) bing used. The returned string uses the "major.minor" format (e.g "1.0").

Parameters:

none

Return Value:

String holding the version number.

Usage:

cOdiRecorder::GetDriveCapacity

Syntax:

```
ulong GetDriveCapacity(int index);
```

Description:

This function is used to query to capacity of a storage device attached to the StreamStor system. The functions returns the total available capacity in bytes. The index is zero based and should be less than the count returned by GetDriveCount.

Parameters:

index - index number of the drive to query for the capacity.

Return Value:

Capacity of the drive in number of bytes.

Usage:

cOdiRecorder::GetDriveCount

Syntax:

```
int GetDriveCount();
```

Description:

This function is used to get the number of drives attached to the StreamStor system.

Parameters:

none

Return Value:

Number of storage devices attached to the StreamStor.

Usage:

cOdiRecorder::GetDriveModel

Syntax:

```
string GetDriveModel(int index);
```

Description:

This function is used to retrieve the model number of a specific drive attached to the StreamStor system. The index is zero based and should be less than the count returned by GetDriveCount.

Parameters:

index - index number of the drive to query for the capacity.

Return Value:

String holding the storage device model number.

Usage:

cOdiRecorder::GetDriveSerial

Syntax:

```
string GetDriveSerial(int index);
```

Description:

This function is used to retrieve the serial number of a specific storage device attached to the StreamStor system. The index is zero based and should be less than the count returned by GetDriveCount.

Parameters:

index - index number of the drive to query for its serial number.

Return Value:

String holding the drive serial number.

Usage:

cOdiRecorder::GetDriveVendor

Syntax:

```
string GetDriveVendor(int index);
```

Description:

This function is used to retrieve the vendor name (manufacturer) a specific storage device attached to the StreamStor system. The index is zero based and should be less than the count returned by GetDriveCount.

Parameters:

index - index number of the drive to query for the vendor name.

Return Value:

String holding the drive vendor name.

Usage:

cOdiRecorder::GetFirmwareVersion

Syntax:

```
string GetFirmwareVersion();
```

Description:

This function is used to retrieve the version number of the currently installed firmware on the StreamStor system. The returned string uses the "major.minor" format (e.g "1.0").

Parameters:

none

Return Value:

String holding the version number.

Usage:

cOdiRecorder::GetFPGAVersion

Syntax:

```
string GetFPGAVersion();
```

Description:

This function is used to retrieve the version number of the software running in the StreamStor system main FPGA. The returned string uses the "major.minor" format (e.g "1.0").

Parameters:

none

Return Value:

String holding the version number.

Usage:

cOdiRecorder::GetFreeCapacity

Syntax:

```
ulong GetFreeCapacity();
```

Description:

This function is used to retrieve the amount of free space (unused space) on the StreamStor system storage devices. The value returned is the number of bytes of space currently available. Note that the StreamStor system has a small percentage of overhead required for recording management so the entire amount reported will not be usable in its entirety for recording storage.

Parameters:

none

Return Value:

Free capacity in number of bytes.

Usage:

cOdiRecorder::GetLength

Syntax:

```
ulong GetLength(string recordingname);
```

Description:

This function is used to retrieve the length of the data captured in the named recording. The amount reported includes the data from all packets in the recording.

Parameters:

recordingname - name of the recording to query.

Return Value:

Number of bytes captured in the recording.

Usage:

cOdiRecorder::GetPacketCount

Syntax:

```
ulong GetPacketCount(string recordingname);
```

Description:

This function is used to retrieve the number of packets captured in a recording.

Parameters:

recordingname - name of the recording to query.

Return Value:

Number of packets captured in the recording.

Usage:

cOdiRecorder::GetPacketErrorCount

Syntax:

```
uint GetPacketErrorCount();
```

Description:

This function is used to retrieve the number of packet transfer errors that occurred in recent record or playback operation. Note that the value is transient and only available for the current session.

Parameters:

none

Return Value:

Number of packet errors that occurred.

Usage:

cOdiRecorder::GetPlayLength

Syntax:

```
ulong GetPlayLength(string recordingname);
```

Description:

This function will return the amount of data played back in the most recent playback operation. The play length is volatile and only valid if a playback has occurred since the last reset or power event.

Parameters:

recordingname - name of the recording to query.

Return Value:

Number of bytes captured in the recording.

Usage:

cOdiRecorder::GetProgress

Syntax:

```
ulong GetProgress();
ulong GetProgress(int recorderindex);
```

Description:

This function will return the amount of data recorded or played during an active recording or playback session. The optional recorderindex is used when there are multiple independent recorders operating simultaneously to allow retrieval of progress amount without requiring a call to cOdiRecorder::SelectedRecorder.

Parameters:

recorderindex – index of recorder (0 based index).

Return Value:

Number of bytes transferred.

Usage:

cOdiRecorder::GetRecorderCount

Syntax:

```
uint GetRecorderCount();
```

Description:

This function is used to get the number of recorders in the system. A chained recorder will have a recorder count that is greater than one. If the recorders in the system are not chained this value indicates the number of independent recorders in the system that can be selected using cOdiRecorder::SelectedRecorder.

Parameters:

none

Return Value:

Number of recorders.

Usage:

cOdiRecorder::GetRecorderName

Syntax:

```
string GetRecorderName();
```

Description:

This function is used to retrieve the name assigned to the StreamStor recorder. A custom name can be assigned to a recorder using the function SetRecorderName.

Parameters:

none

Return Value:

Name of the recorder.

Usage:

cOdiRecorder::GetRecordingCount

Syntax:

```
int GetRecordingCount()
```

Description:

This function returns the number of recordings on the StreamStor system. the Recordings are also available in an enumerable list "Recordings".

Parameters:

none

Return Value:

Number of recordings.

See Also:

Usage:

cOdiRecorder::GetRecordingName

Syntax:

```
string GetRecordingName(int index)
```

Description:

This function returns the name of a recording by index. The index starts at zero(0) and must be less than the count returned by GetRecordingCount. The recording names are also available in an enumerable list "Recordings[].Name".

Parameters:

index - index of recording to get name.

Return Value:

String with recording name.

See Also:

Usage:

cOdiRecorder::GetRecordingFileSize

Syntax:

```
ulong GetRecordingFileSize(string recordingname)
```

Description:

This function returns the overall size of a recordings file. Note that this is not the length of the recorded data. The size of the file defines the limits of how much data a recording can hold.

Parameters:

recordingname - recording name to retieve file size.

Return Value:

Size of recording file in bytes.

See Also:

Usage:

cOdiRecorder::GetSystemErrorCode

Syntax:

```
uint GetSystemErrorCode();
```

Description:

When a system error has occurred as indicated by the IsSystemErrorSet function call, this function can be used to retrieve an error code to help Conduant support in diagnosing the problem. A system error is generally an unexpected fault in the operation of the StreamStor hardware, storage devices or software.

Parameters:

none

Return Value:

The system error code.

Usage:

cOdiRecorder::GetTotalCapacity

Syntax:

```
ulong GetTotalCapacity();
```

Description:

This function is used to retrieve the total storage capacity on the StreamStor system. This includes both used space and free space. The total capacity may not equal the sum of the drive capacity amounts returned by GetDriveCapacity due to file system and other overhead not accounted for in drive capacity reporting.
Note that the total capacity doe not take into account overhead incurred by each recording such that the sum of all recorded data when full will be less than the total capacity.

Parameters:

none.

Return Value:

Storage capacity of system (bytes).

Usage:

cOdiRecorder::IsBufferOverflowed

Syntax:

```
bool IsBufferOverflowed();
```

Description:

If this function returns true, the system has run into a buffer overflow condition. A buffer overflow generally occurs when the incoming data rate is higher than the rate of writing to storage. A properly defined system should never see this occur in normal operation.

Parameters:

none.

Return Value:

True if a buffer overflow as occurred, False otherwise.

Usage:

cOdiRecorder::IsPlaying

Syntax:

```
bool IsPlaying();
```

Description:

This function can be used to indicate if the system is currently in playback of a recording. If a playback ends due to the programmed playback size being reached, this function will return false to indicate that the playback has ended.

Parameters:

none.

Return Value:

True if playback in progress, False otherwise.

Usage:

cOdiRecorder::IsRecording

Syntax:

```
bool IsRecording();
```

Description:

If this function returns true, the system is currently recording data. If programmed to wrap, the record operation will never exit and this function will always return true.

Parameters:

none.

Return Value:

True if recording is in progress, False otherwise.

Usage:

cOdiRecorder::IsRecordingOpen

Syntax:

```
bool IsRecordingOpen();
```

Description:

If this function returns true, there is a recording currently open for reading or writing. Any open recording must be closed before a playback or record operation can be started.

Parameters:

none.

Return Value:

True if a recording is open, False otherwise.

Usage:

cOdiRecorder::IsRecordingOpenForRead

Syntax:

```
bool IsRecordingOpenForRead();
```

Description:

If this function returns true, there is a recording currently open for reading. Any open recording must be closed before a playback or record operation can be started.

Parameters:

none.

Return Value:

True if a recording is open for read, False otherwise.

Usage:

cOdiRecorder::IsRecordingOpenForWrite

Syntax:

```
bool IsRecordingOpenForWrite();
```

Description:

If this function returns true, there is a recording currently open for writing. Any open recording must be closed before a playback or record operation can be started.

Parameters:

none.

Return Value:

True if a recording is open for write, False otherwise.

Usage:

cOdiRecorder::IsStorageOverflowed

Syntax:

```
bool IsStorageOverflowed();
```

Description:

If this function returns true, the current or last recording reached the storage capacity limits. A recording started with wrap enabled will never reach this state.

Parameters:

none.

Return Value:

True if a recording has used all available storage, False otherwise.

Usage:

cOdiRecorder::IsSystemErrorSet

Syntax:

```
bool IsSystemErrorSet();
```

Description:

If this function returns true, a critical system error has occurred. System errors indicate an unexpected hardware, storage, or software problem has occurred.

Parameters:

none.

Return Value:

True if a system error has occurred, False otherwise.

Usage:

cOdiRecorder::IsUnitAttentionSet

Syntax:

```
bool IsUnitAttentionSet();
```

Description:

If this function returns true, a unit attention event has occurred on the system. This is an indication that a power cycle or reset has occurred on the StreamStor system since the session was started.

Parameters:

none.

Return Value:

True if unit attention is active, False otherwise.

Usage:

cOdiRecorder::IsValidRecordingName

Syntax:

```
bool IsValidRecordingName(string checkname);
```

Description:

If this function returns true, the provided name is a valid recording on the system. If the function returns false, the name is not valid. Provide a way to validate a recording name before using in other functions.

Parameters:

`string checkname` - Reoording name to check.

Return Value:

True if name valid, False otherwise.

Usage:

cOdiRecorder::OpenRecording

Syntax:

```
void OpenRecording(string recordingname, bool forRead)
```

Description:

This function opens a recording for reading or writing of packets. A recording must be opened before packets can be written or read from it. The recording must also be closed using CloseRecording before record, playback and other operations can proceed.

Parameters:

`recordingname` - Name of recording to open.
`forRead` - determines if recording is being opened for read (true) or write (false).

Return Value:

none

See Also:

Usage:

cOdiRecorder::Playback

Syntax:

```
void Playback(string recordingname)
```

Description:

This function starts a playback of a recording to the ODI2 port. Playback continues until all data has been played or Stop is called.

Parameters:

`recordingname` - Name of recording to playback.

Return Value:

none

See Also:

Usage:

cOdiRecorder::Playloop

Syntax:

```
void Playloop(string recordingname)
void Playloop(string recordingname, uint loops)
```

Description:

This function starts a continuous (infinite) playback of a recording to the ODI2 port. When no "loops" parameter is provided the playback will continue until the Stop function is called. When "loops" is provided the system will playback the recording that number of times and stop automatically.

Parameters:

`recordingname` - Name of recording to playback.
`loops` - number of times to play the recording in a continuous loop.

Return Value:

none

See Also:

Usage:

cOdiRecorder::ReadPackets

Syntax:

```
int ReadPackets(ulong startpacket, array<byte>^ buffer, int
numberOfPackets)
```

Description:

This function requires a previously opened recording (for read). The function will move packets from the open recording to a user defined buffer. The offset defines the starting point in the recorded data and the length defines the requested amount of data to read. If the amount of data available is less than length, the actual amount read will be returned in the length parameter.

Parameters:

`packet` - packet number to read (packet numbers start at 1)
`buffer` - user defined buffer to hold the data.
`numberOfPackets` - number of packets to return

Return Value:

Number of packets actually returned into buffer. May be less than requested if end of data is reached.

See Also:

Usage:

cOdiRecorder::ReadPacket

Syntax:

```
int ReadPacket(ulong packet, array<byte>^ buffer)
```

Description:

This function requires a previously opened recording (for read). The function will move a packet from the open recording to a user defined buffer. The offset defines the packet in the recorded data (packet numbering starts at 1). If the buffer is not large enoughamount of data available is less than length, the actual amount read will be returned in the length parameter.

Parameters:

`packet` - packet number to read (packet numbers start at 1)
`buffer` - user defined buffer to hold the data.

Return Value:

Size of packet

See Also:

Usage:

cOdiRecorder::Reconfigure

Syntax:

```
void Reconfigure(string configname)
```

Description:

This function resets and restarts the StreamStor system software. The configname is used during the reboot in place of the default configuration file or the previously loaded configuration. The configuration file named must already exist on the recording system. Configuration files include parameters that define which disk drives will be used to configure the recorder. This allows a system to be re-configured "on the fly" to use a different set of disks.

Parameters:

`configname` - name of the configuration file to use when re-booting.

Return Value:

none

See Also:

Usage:

cOdiRecorder::Record

Syntax:

```
void Record(string recordingname)
```

Description:

This function starts a recording on the StreamStor system. The recording will continue until the recording file storage space is exhausted or cOdiRecorder::Stop is called.

Parameters:

`recordingname` - name of recording to use for this recording (must be empty).

Return Value:

none

See Also:

Usage:

cOdiRecorder::RenameRecording

Syntax:

```
void RenameRecording(string currentname, string newname)
```

Description:

This function can be used to rename an existing recording on the StreamStor system.

Parameters:

`currentname` - name of an existing recording.
`newname` - new name for this recording.

Return Value:

none

See Also:

Usage:

cOdiRecorder::SelectedRecorder

Syntax:

```
void SelectedRecorder(int recorderindex)
int SelectedRecorder()
```

Description:

This function can be used to set or retrieve the currently selected recorder. This function is primarily for use when multiple independent recorders are configured on the system and allows the selection of the recorder to control.

Parameters:

`recorderindex` – A 0 based index of the recorder to select. Must be less than the number of recorders.

Return Value:

Currently selected recorder index (0 based).

See Also:

Usage:

cOdiRecorder::SetPassThru

Syntax:

```
void SetPassThru(bool AllowPassThru)
```

Description:

This function can be used to enable Interlaken (Odi) pass-thru of the incoming data stream on the ODI1 port to the ODI2 port. Pass-thru must be disabled to allow recording of data from Odi.

Parameters:

`AllowPassThru` - when "true" the pass-thru will be enabled. When "false" the pass-thru will be disabled.

Return Value:

none

See Also:

Usage:

cOdiRecorder::SetRecorderName

Syntax:

```
void SetRecorderName(string recordername)
```

Description:

This function assign a user defined name to the StreamStor system. If no name is assigned a default name is used. The name can be retrieved using the GetRecorderName function.

Parameters:

`recordername` - name to use for recorder.

Return Value:

none

See Also:

Usage:

cOdiRecorder::Stop

Syntax:

```
void Stop()
```

Description:

This function can be used to stop an in progress record or playback operation. It should not be used when reading or writing to an open recording.

Parameters:

none

Return Value:

none

See Also:

Usage:

cOdiRecorder::TruncateRecording

Syntax:

```
void TruncateRecording(string recordingname)
```

Description:

This function will reduce the size of a recording to fit the amount of data actually recorded.

Parameters:

`recordingname` - name of recording to truncate.

Return Value:

none

See Also:

Usage:

cOdiRecorder::WritePacket

Syntax:

```
int WritePacket(array<byte>^ buffer, uint size)
```

Description:

This function is used to write a packet to a recording that has been opened for write. Packets are always appended to the end of the recording. The returned value is the number of bytes written and should always be checked to insure the end of the recording file has not been reached. If writing the packet to the recording will result in exceeding the recording space, the size will return as 0 and the packet is not written to the recording.

Parameters:

buffer - user allocated buffer holding the packet to write to the recording.
size - size of the packet.

Return Value:

Number of bytes written (usually the same as size).

See Also:

Usage:

Class cOdi

Namespace: `Conduant.StreamStor3`
Assembly: ssapi3.dll
Members
`cOdiPorts^ Ports;`

Class cOdiPorts

Namespace: `Conduant.StreamStor3`
Assembly: ssapi3.dll

Members

`Uint32 Count;`
Holds the number of Odi ports on the recorder.

`IList<string>^ Name;`
List of port names available on the recorder ("ODI1", "ODI2").

`Item[string Name];`
ReadOnly collection of ports, indexed by port name.

`string Item[Name].Name;`
Name of port.

`PortCapability Item[Name].GetCapability();`
Return information about the capabilites of the port.

`void Item[Name].Activate();`
Activate the port, turn on transmitters and receivers

`void Item[Name].Deactivate();`
Deactivate the port, turn off transmitters and receivers

Enums

Namespace: Conduant.StreamStor3
Assembly: ssapi3.dll

```
public enum class LaneRate
{
   RATE_12R5G = 1,
   RATE_14R1G = 2
};

public enum class FlowControl
{
   NONE = 1,
   INBAND = 2,
   OUTOFBAND_1WIRE = 3,
   OUTOFBAND_BACKPLANE_0 = 4, // PXI_TRIG0 or AXIe TRIG0
   OUTOFBAND_BACKPLANE_1 = 5, // PXI_TRIG1 or AXIe TRIG1
   OUTOFBAND_BACKPLANE_2 = 6, // PXI_TRIG2 or AXIe TRIG2
   OUTOFBAND_BACKPLANE_3 = 7, // PXI_TRIG3 or AXIe TRIG3
   OUTOFBAND_BACKPLANE_4 = 8, // PXI_TRIG4 or AXIe TRIG4
   OUTOFBAND_BACKPLANE_5 = 9, // PXI_TRIG5 or AXIe TRIG5
   OUTOFBAND_BACKPLANE_6 = 10, // PXI_TRIG6 or AXIe TRIG6
   OUTOFBAND_BACKPLANE_7 = 11, // PXI_TRIG7 or AXIe TRIG7
   OUTOFBAND_BACKPLANE_8 = 12, // PXI_TRIG8 or AXIe TRIG8
   OUTOFBAND_BACKPLANE_9 = 13, // AXIe TRIG9
   OUTOFBAND_BACKPLANE_10 = 14, // AXIe TRIG10
   OUTOFBAND_BACKPLANE_11 = 15, // AXIe TRIG11
   OUTOFBAND_BACKPLANE_12 = 16 // AXIe TRIG12
};
```

Class PortCapability

Namespace: `Conduant.StreamStor3`
Assembly: ssapi3.dll

Members

```
string Name;
string Version; // Version of the standard supported
LaneRate[] LaneRates; // list of supported lane rates.
int[] Lanes; // list of supported lane widths.
int[] TxBurstMaxes; // Transmitter BurstMax supported (bytes)
int RxBurstMax; // Receiver BurstMax in bytes.
FlowControl[] FlowControls; // Flow control methods supported
int ChannelMax; // Maximum Interlaken channels supported
bool TxRateMatching; // Interlaken rate matching supported
```

Class PortStatistics

Namespace: `Conduant.StreamStor3`
Assembly: ssapi3.dll

Members

```
long BytesSent; // cumulative bytes sent
long BytesReceived; // cumulative bytes received
long BadBurstsReceived; // number of bursts received with bad CRC
long TxFlowControlHoldoffs; // number of clock cycles TX held off
```

Enums

PortStatus

```
public enum PortStatus
{
   ACTIVE = 0x0001, // port has been activated
   TX_READY = 0x0002, // Ready to transmit and flow control allows
   RX_READY = 0x0004, // Rx synchronized and aligned
   RX_LANE_ERROR = 0x0008, // RX error in metaframe, scrambler, …
   RX_BURST_MAX_ERROR = 0x0010, // Received too large of a burst.
   RX_CRC_ERROR = 0x0020, // CRC error in Burst
   RX_OVERRUN = 0x0040,
   RX_FC_STATUS0 = 0x00010000, // Received flow control status bit
0
   RX_FC_STATUS1 = 0x00020000,
   RX_FC_STATUS2 = 0x00040000,
   RX_FC_STATUS3 = 0x00080000,
};
```

Chapter 9
Technical Support

Contact information:
Telephone: 1-303-485-2721
Email: support@conduant.com
Website: https://zendesk.conduant.com

Conduant wants to be sure that your StreamStor system works correctly and stays working correctly. In the event, however, that you are unable to get your new system to work properly, or if a working system ceases to function, we will do all that we can to get your system back online.

Solving the problem is largely a matter of data collection and steps that must be taken one at a time. In order for us to better serve you, we ask that you take the time to perform some troubleshooting steps prior to calling us. This way, you can provide us with the most meaningful information possible that will help us solve the problem.

Is the problem one that obviously requires replacement parts due to physical damage to the system? If yes, then please gather the information described below and report the problem to tech support through the Conduant support website.

Have you confirmed that no cabling has been inadvertently disconnected or damaged while working around the equipment?

Are all cards properly seated in the PXIe slots?

Has the software installation been corrupted? Have you tried re-installing software?

Have you checked the Conduant web site for any technical bulletins?

Have you recently installed a new compiler or a new Windows Service Pack / Update?

If you are unable to resolve the problem on your own, then please visit our support website and open a support ticket. To open a support ticket, go to conduant.com (click support) or conduant.zendesk.com. Click on “Submit a request” or “Sign in”. You can create an account on the support website to allow management of your outstanding issues and see all responses and updates in one place.

The more information that you can provide will help to get your issue resolved quickly. Please provide at least the following information:

- StreamStor (Cobra) board serial number
- Software versions
- Configuration information (# of drives, etc.)
- Description of 3rd party equipment (digitizers, etc.)
- Host computer model and OS

We will do all that we can to resolve the problem as quickly as possible.

Contacting Technical Support

E-mail: support@conduant.com

Phone: 1.303.485.2721

Fax: 1.303.485.5104

Web: zendesk.conduant.com

Mail: Conduant Corporation
Technical Support
1501 S SUNSET ST STE D
LONGMONT, CO 80501-6757

Note: If returning equipment please contact us first using the support website and use the ticket number assigned as an RMA number on your packaging so we can track your hardware.