Addendum for VLBI Users

Copyright and Trademarks

Printed in the United States.

Publication date: December 9, 2005

Table of Contents

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.

About This Manual

This addendum describes API functions that are primarily of interest to the VLBI community.

XLRClearOption

Syntax:

```
XLR_RETURN_CODE XLRClearOption( SSHANDLE xlrDevice,
   UINT options_to_clear )
```

Description:

`XLRClearOption` clears an option previously set by `XLRSetOption`, or clears all options. When an option is cleared, it is set to its default value. See `XLRSetOption` for the list of available options and default values. To clear an option, the drives must be idle (i.e., no recording or playback in progress).

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`options_to_clear`* is a vector of options to clear.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE              xlrDevice;
XLR_RETURN_CODE       xlrStatus;

//
// This example shows how to set options to their default values
// and how to set and clear a specific option.
//

xlrStatus = XLROpen( 1, &xlrDevice );

// Set all options to their default values.
xlrStatus = XLRClearOption( device, SS_ALL_OPTIONS );

//
// Set the desired option.  In this example, set the option
// to reserve a block of space.
//
xlrStatus = XLRSetOption( xlrDevice, SS_OPT_RESERVE_END_BLOCK );
    . . . record some data . . .

// Clear the reserve block option.
xlrStatus = XLRSetClear( xlrDevice, SS_OPT_RESERVE_END_BLOCK );
```

See Also:

XLRSetOption and XLRGetOption.

XLRDiskRepBlkCount

Syntax:

```
DWORD XLRDiskRepBlkCount( SSHANDLE xlrDevice, UINT Bus, UINT
MasterSlave )
```

Description:

`XLRDiskRepBlkCount` returns the replaced block count for the specified bus and drive. The replaced block count is the number of blocks that have been replaced during a playback when fill data was used to replace unreadable data.

The replaced block counter(s) will be reset on a read or playback request if:

- the request is non-sequential from a previous read/playback or
- the previous read/playback ended due to reaching a play limit request (see `XLRSetPlaybackLength`) or
- there have been intervening commands that accessed the disk, such as a record.

In other words, the counter(s) are not reset if a play (or read) is followed by a subsequent play (or read) that is sequential to the previous one and play limit is not used.

If the StreamStor is in bank mode, this command will return the number of replaced blocks for the drive in the selected bank.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`bus`* is the ATA bus number of the drive.
- *`MasterSlave`* is `XLR_MASTER_DRIVE` (0) or `XLR_SLAVE_DRIVE` (1) to select the master or the slave drive on the ATA bus.

Return Value:

The replacement block count for the specified drive. If no blocks are replaced or an error occurs, zero is returned.

Usage:

```
//
// This example shows FPDP playback using a fill pattern.  After
// playback stops, the disk replacement block count is retrieved.
//

SHANDLE            xlrDevice;
DWORD              repBlockCount;
XLR_RETURN_CODE    xlrStatus;

      ...
xlrStatus = XLROpen( 1, &xlrDevice );

//
// To use fill data, you must first set the skip check dir option.
//
xlrStatus = XLRSetOption( xlrDevice, SS_OPT_SKIPCHECKDIR );

//
// Establish the fill pattern to use for blocks that cannot be read.
// In this example, the fill pattern is all zeros, though you can
// use whatever pattern you like.
//
xlrStatus = XLRSetFillData( xlrDevice, 0x00000000 );

// Set the mode to playback over the external port.
xlrStatus = XLRSetMode( xlrDevice,SS_MODE_EXT );

xlrStatus = XLRSetFPDPMode( xlrDevice,SS_FPDP_XMIT,0 );

// Start playback at the beginning of the recording.
xlrStatus = XLRPlayback( xlrDevice, 0, 0 );

     . . . play back data . . .

XLRStop( xlrDevice );

// Get the number of replaced blocks on the master on Bus 2.
repBlockCount =
   XLRDiskRepBlkCount( xlrDevice, 2, XLR_MASTER_DRIVE );
```

See Also:

XLRTotalRepBlkCount, XLRSetFillData, XLRPlayback, XLRSetOption, XLRSetBankMode and XLRSelectBank.

XLRGetDiskLength

Syntax:

```
DWORDLONG XLRGetDiskLength( SSHANDLE xlrDevice, UINT Bus,
   UINT MasterSlave )
```

Description:

`XLRGetDiskLength` returns the length (in bytes) of the data recorded on the specified drive. The length is rounded to the nearest 64-kilobyte boundary. To get the length of a drive, the drives must be idle (i.e., no recording or playback in progress).

If the StreamStor is in bank mode, this command will return the length of data on the specified drive in the selected bank.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`bus`* is the ATA bus number of the drive.
- *`MasterSlave`* is `XLR_MASTER_DRIVE` (0) or `XLR_SLAVE_DRIVE` (1) to select the master or the slave drive on the ATA bus.

Return Value:

Length (in bytes) of data on the specified drive, rounded to the nearest 64-kilobyte boundary. If an error occurs, the returned value is `SS_UNDEFINED_LENGTH`.

Usage:

```
SSHANDLE           xlrDevice;
DWORDLONG          diskLength;
XLR_ERROR_CODE     xlrErrorCode;
XLR_RETURN_CODE    xlrStatus;

xlrStatus = XLROpen( 1, &xlrDevice );

// Get the length of data on the slave drive on bus 0.
diskLength = XLRGetDiskLength( xlrDevice, 0, XLR_SLAVE_DRIVE );
if( disklength == SS_UNDEFINED_LENGTH ){
   xlrErrorCode = XLRGetLastError();
   printf( "GetDiskLength failed.  Error = %lu\n", xlrErrorCode );
}
else {
   printf( "Length of data on Slave drive on bus zero = %llu\n",
     diskLength );
}
```

XLRGetDriveStats

Syntax:

```
XLR_RETURN_CODE XLRSetDriveStats( SSHANDLE xlrDevice, UINT Bus,
   UINT MasterSlave, S_DRIVESTATS dstats )
```

Description

XLRGetDriveStats retrieves drive statistics that have been collected by StreamStor while the SS_OPT_DRVSTATS option is on (see XLRSetOption). To retrieve drive statistics, the drives must be idle (i.e., no recording or playback in progress).

Parameters:

- *xlrDevice* is the device handle returned from a previous call to XLROpen.
- *bus* is the ATA bus number of the drive.
- *masterSlave* is XLR_MASTER_DRIVE (0) or XLR_SLAVE_DRIVE (1) to select the master or the slave drive on the ATA bus.
- *dstats* is an array of structures of type S_DRIVESTATS. The array will receive the drive statistics for the requested master (or slave) drive on *bus*.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
//
// This example shows how to retrieve drive statistics that were
// collected during a playback of data.
//

SSHANDLE          xlrDevice;
XLR_RETURN_CODE   xlrStatus;
S_DRIVESTATS      dstats[XLR_MAXBINS];
UINT              binNum;

xlrStatus = XLROpen( 1, &xlrDevice );

//
// Set the option so that drive statistics will be
// collected during StreamStor read and write operations.
//
xlrStatus =  XLRSetOption( xlrDevice, SS_OPT_DRVSTATS );

//Start some playback
```

```
xlrStatus = XLRPlayback( xlrDevice, 0, 0 );

      . . . continue playback . . .
//
// The StreamStor must be idle in order to retrieve the
// drive statistics.
//
XLRStop( xlrDevice );

// Get the statistics for the master drive on bus 0.
xlrStatus = XLRGetDriveStats( xlrDevice, 0, XLR_MASTER_DRIVE, dstats );

for( binNum=0; binNum<XLR_MAXBINS; binNum++ )
{
   printf( "  Count for bin[%d] = %lu\n", binNum, dstats[binNum].count );
}
```

See Also:

XLRSetOption, XLRClearOption, and XLRSetDriveStats.

XLRGetOption

Syntax:

```
XLR_RETURN_CODE XLRGetOption( SSHANDLE xlrDevice, UINT
options_to_get, PBOOLEAN options_on )
```

Description

`XLRGetOption` is used to determine if one or more options are set.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`options_to_get`* is a vector of options to query.
- *`options_on`* is the returned `BOOLEAN` indicating if all the options in *`options_to_get`* are set. If `TRUE`, all of the options in *`options_to_get`* are set. If set to `FALSE`, one or more of the options in *`options_to_get`* are not set.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE              xlrDevice;
XLR_RETURN_CODE       xlrStatus;
BOOLEAN               options_on;

xlrStatus = XLROpen( 1, &xlrDevice );

// See if drive statistics are getting collected.
xlrStatus = XLRGetOption( device, SS_OPT_DRVSTAT, &options_on );
if ( options_on == TRUE )
{
   printf ( "Drive statistics are getting collected.\n" );
}

// See if drive stats and skip check dir options are set.

xlrStatus = XLRSetOption( xlrDevice, SS_OPT_DRVSTATS |
     SS_OPT_SKIPCHECKDIR, &options_on );
```

```
if( options_on == TRUE )
{
   printf
    ( "Drive stats are getting collected and skip check dir is set.\n" );
}
else
{
   printf( "Drive stats is not set or skip check dir is not set.\n" );
}
```

See Also:

XLRSetOption and XLRGetOption.

XLRGetPlayBufferStatus

Syntax:

```
XLR_RETURN_CODE XLRGetPlayBufferStatus( SSHANDLE xlrDevice, PUINT
status )
```

Description

`XLRGetPlayBufferStatus` returns the status of the playback buffer. The playback buffer is used when the playback trigger is armed. See the `SS_OPT_PLAYARM` option of the `XLRSetOption` command.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`status`* is a pointer to a `UINT` (unsigned integer) that will be written with the status of the play buffer. The playback buffer will be in one of the following states:
  - `SS_PBS_IDLE` – The playback buffer is not in use.
  - `SS_PBS_FULL` – The playback buffer is full and a playback is not in progress.
  - `SS_PBS_FILLING` – Data is streaming into the playback buffer.
  - `SS_PBS_PLAYING` – Data from the playback buffer is playing.
  - `SS_PBS_UNKNOWN` – The status of the playback buffer cannot be determined.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
//
// Example of using FPDP to playback and then retrive the playback buffer
// status.
//

SSHANDLE          xlrDevice;
XLR_RETURN_CODE   xlrStatus;
UINT              playBuffStatus;

XLROpen( 1, &xlrDevice );

// Use the external port.
xlrStatus = XLRSetMode( xlrDevice, SS_MODE_EXT );

// Use FPDP mode transmit for playback.
xlrStatus = XLRSetFPDPMode( xlrDevice, S_FPDP_XMIT, 0 );

// Prepare for playback.
xlrStatus = XLRSetOption( xlrDevice, SS_OPT_PLAYARM );

//
//Since the playback buffer is not in use yet, the status returned
//here should be SS_PBS_IDLE.
//
xlrStatus = XlrGetPlayBufferStatus( xlrDevice, &playBuffStatus);

//
// Since SS_OPT_PLAYARM has been set, this call to XLRPlayback
// will not cause the data to flow yet.
//
xlrStatus = XLRPlayback( xlrDevice, 0, 0 );

//
// XLRPlayback has been called, but XLRPlayTrigger has not.  The status
// of the playback buffer should be SS_PBS_FILLING now.
//
xlrStatus = XlrGetPlayBufferStatus( xlrDevice, &playBuffStatus);

//
// Periodically call XlrGetPlayBufferStatus until the status of
// the playback buffer is SS_PBS_FULL.
//
            . . .
//
// Buffer is now full, so call XLRPlayTrigger to start the data
// flowing out of the buffer.
xlrStatus = XLRPlayTrigger( xlrDevice );

 ... playback the desired length of time . . .

// Stop the playback.
```

```
XLRStop( xlrDevice );
```

See Also:

XLRPlayTrigger, XLRSetOption, and XLRPlayBack.

XLRPlayTrigger

Syntax:

```
XLR_RETURN_CODE XLRPlayTrigger( SSHANDLE xlrDevice )
```

Description

`XLRPlayTrigger` starts read data flowing.

Typically, when `XLRPlayback` is called, preparations are made internally to begin playback. When preparations are complete, the data begins to flow. So, there is a delay between the call to `XLRPlayback` and the flow of data. To minimize the delay, you can call `XLRSetOption` to set the `SS_OPT_PLAYARM` option. The causes the preparations for playback to be made, but playback does not start until `XLRPlayTrigger` is called.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
// Example using FPDP.

SSHANDLE           xlrDevice;
XLR_RETURN_CODE    xlrStatus;

XLROpen( 1, &xlrDevice );

// Use the external port.
xlrStatus = XLRSetMode( xlrDevice, SS_MODE_EXT );

// Use FPDP mode transmit for playback.
xlrStatus = XLRSetFPDPMode( xlrDevice, S_FPDP_XMIT, 0 );

// Prepare for playback.
xlrStatus = XLRSetOption( xlrDevice, SS_OPT_PLAYARM );

//
// Since SS_OPT_PLAYARM has been set, this call to XLRPlayback
// will not cause the data to flow yet.
//
xlrStatus = XLRPlayback( xlrDevice, 0, 0 );

//
// If we do not sleep here, then the playback will work as
```

```
// it does ordinarily.  If instead we sleep, the internal
// playback buffer will fill up.  This should take no
// longer than three seconds.
//
 ... sleep, allowing playback buffer to fill ...

//
// The playback buffer should now be full.  Start the data flowing
// out of the buffer.
//
xlrStatus = XLRPlayTrigger( xlrDevice );

 ... playback the desired length of time . . .

// Stop the playback.
XLRStop( xlrDevice );
```

See Also:

XLRSetOption and XLRPlayback.

XLRSetFillData

Syntax:

```
XLR_RETURN_CODE XLRSetFillData( SSHANDLE xlrDevice, UINT pattern )
```

Description

`XLRSetFillData` sets the fill data pattern to the requested pattern. The fill data pattern can be output on playback to signal that the output stream from the disk is unable to meet the requested bandwidth. Calling `XLRSetFillData` sets the pattern and enables this functionality.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`pattern`* is the fill pattern to use.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
//
// This example shows playback using FPDP with a fill
// pattern.
//
SHANDLE            xlrDevice;
XLR_RETURN_CODE    xlrStatus;

xlrStatus = XLROpen( 1, &xlrDevice );

//
// To use fill data, you must first set the skip check dir option.
//
xlrStatus = XLRSetOption( xlrDevice, SS_OPT_SKIPCHECKDIR );

//
// Establish the fill pattern to use for blocks that cannot be read.
// In this example, the fill pattern is all zeros.
//
xlrStatus = XLRSetFillData( xlrDevice, 0x00000000 );

// Set up to playback over the external port.
xlrStatus = XLRSetMode( xlrDevice,SS_MODE_EXT );

xlrStatus = XLRSetFPDPMode( xlrDevice,SS_FPDP_XMIT,0 );
```

```
//
// Start playback at the beginning of the recording, filling in
// any unreadable data with the fill pattern.
//
xlrStatus = XLRPlayback( xlrDevice, 0, 0 );

     . . . play back data . . .

XLRStop( xlrDevice );
```

See Also:

XLRSetOption and XLRPlayback.

XLRSetDriveStats

Syntax:

```
XLR_RETURN_CODE XLRSetDriveStats( SSHANDLE xlrDevice, S_DRIVESTATS
dstats )
```

Description

`XLRSetDriveStats` customizes the bin ranges to be used when collecting drive statistics.

By default, drive statistics are not collected. To start collection of drive statistics, you must first call `XLRSetOption` with the `SS_OPT_DRVSTATS` option. When `XLRSetOption` is called, the bin ranges are set to their default values (see the structure definition of `S_DRIVESTATS`). If desired, you can customize the upper bound of the bin ranges that are used to collect the statistics. To do so, you populate a `S_DRIVESTATS` structure with the desired upper bounds and then call `XLRSetDriveStats`.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`dstats`* is an `S_DRIVESTATS` structure that contains the upper bounds for the bins.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
// Example of setting drive statistics bin ranges.

SSHANDLE        xlrDevice;
XLR_RETURN_CODE xlrStatus;
S_DRIVESTATS    dstats;

XLROpen( 1, &xlrDevice );

// Set the option to collect drive statistics.
xlrStatus = XLRSetOption( xlrDevice, SS_OPT_DRVSTATS );

// Instead of using the default upper bounds for drive
// statistics ranges, use these.
//
dstats[0].range =  75000;
dstats[1].range = 100000;
dstats[2].range = 105000;
dstats[3].range = 200000;
dstats[4].range = 205000;
dstats[5].range = 300000;
dstats[6].range = 305000;
dstats[7].range = ULONG_MAX;

//Set the new bin ranges.
xlrStatus =  XLRSetDriveStats( xlrDevice, dstats );
```

See Also:

XLRSetOption, XLRClearOption, and XLRGetDriveStats.

XLRSetOption

Syntax:

```
XLR_RETURN_CODE XLRSetOption( SSHANDLE xlrDevice,
   UINT options_to_set )
```

Description

`XLRSetOption` sets one or more options. To set an option, the drives must be idle (i.e., no recording or playback in progress).

The options are:

- `SS_OPT_DRVSTATS` – When set, drive statistics are collected. Otherwise, statistics are not collected. By default, this option is not set. See `XLRGetDriveStats`.
- `SS_OPT_PLAYARM` - When set, the StreamStor is armed for a two-stage playback. You set this option and then call `XLRPlayback`. Data will be buffered up for playback, but no data will play until triggered. (See `XLRPlayTrigger.`) By default, this option is not set.
- `SS_OPT_SKIPCHECKDIR` - When set, the directory sanity check that is ordinarily done by most XLR commands is not done. This is necessary to playback data from a set of disks where one or more disks are unreadable because of a damaged or missing disk. To use, you set `SS_OPT_SKIPCHECKDIR` and then call `XLRSetFillData` with pattern to substitute for unreadable data. During playback, the unreadable data will be replaced with the pattern set by `XLRSetFillData`. By default, this option is not set. See also `XLRSetFillData` and `XLRDiskRepBlkCount`.
- `SS_OPT_REALTIMEPLAYBACK` – "Real-time playback" is automatically enabled by calling `XLRSetFillData`. To turn real-time playback off, unset this option.
- `SS_OPT_RESERVE_END_BLOCK` - When set, the StreamStor stops recording before the StreamStor is full (or, in the case of bank mode, before the selected bank is full). When set, the available capacity for recording is:
  - Bank mode:
    available capacity = drive capacity –
    (4 * 196584 * number of drives in a module)
  - Non-bank mode:
    available capacity = drive capacity - (4 * 196584 * total number of drives)

When `XLROpen` is called, each option is set to its default value.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`options_to_set`* is a vector containing one or more of the above options.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE              xlrDevice;
XLR_RETURN_CODE       xlrStatus;

xlrStatus = XLROpen( 1, &xlrDevice );

//
// Set the desired option.  In this example, set the option
// to reserve a block of space.
//
xlrStatus = XLRSetOption( xlrDevice, SS_OPT_RESERVE_END_BLOCK );
    . . . record some data . . .

// Clear the reserve block option.
xlrStatus = XLRClearOption( xlrDevice, SS_OPT_RESERVE_END_BLOCK );
```

See Also:

XLRClearOption and XLRGetOption.

XLRSkip

Syntax:

```
UINT XLRSkip( SSHANDLE xlrDevice, UINT skipLength, BOOLEAN forward
)
```

Description

`XLRSkip` is used during `XLRPlayback` to skip forward and back the requested number of bytes.

To accomplish a skip, StreamStor buffers some playback data internally. It cannot skip forward or backwards more than the number of bytes in the internal buffer. If `XLRSkip` is called with `skipLength` greater than the number of bytes in the buffer, it will skip the maximum number of bytes available in the buffer. Otherwise, it will skip `skipLength` bytes.

`XLRSkip` can only be used while playback is through the external port.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`skipLength`* is the number of bytes to skip. The number of bytes to skip must be 8 byte aligned (i.e., the number must be a multiple of 8).
- *`forward`* is the skip direction. If forward is `TRUE`, playback will skip forward *`skipLength`* bytes. Otherwise, playback will skip backwards *`skipLength`* bytes.

Return Value:

The number of bytes that were skipped.

Usage:

```
SHANDLE            xlrDevice;
UINT               bytesActuallySkipped=0;
UINT               bytesToSkip=0;
XLR_RETURN_CODE    xlrStatus=XLR_FAIL;

xlrStatus = XLROpen( 1, &xlrDevice );

// Set up to playback over the external port.
xlrStatus = XLRSetMode( xlrDevice,SS_MODE_EXT );

// Set the FPDP mode for transmitting (playback) over FPDP.
xlrStatus = XLRSetFPDPMode( xlrDevice,SS_FPDP_XMIT,0 );

// Start playback at the beginning of the recording.
xlrStatus = XLRPlayback( xlrDevice, 0, 0 );

     . . . play back data . . .

// Skip ahead over some of the data.
bytesToSkip = 1024;
bytesActuallySkipped = XLRSkip( xlrDevice, bytesToSkip, TRUE );

     . . . data is skipped and continues to play . . .

// Skip backwards.
bytesToSkip = 4098;
bytesActuallySkipped = XLRSkip( xlrDevice, bytesToSkip, TRUE );

     . . . data is skipped and continues to play . . .

// Stop playback.
XLRStop( xlrDevice );
```

See Also:

XLRSetMode and XLRPlayback.

XLRTotalRepBlkCount

Syntax:

```
DWORD XLRTotalRepBlkCount( SSHANDLE xlrDevice )
```

Description

`XLRTotalRepBlkCount` returns the total number of blocks that were replaced during a playback. The replaced block count is the number of blocks that have been replaced during a playback when fill data is used to replace unreadable data.

The replaced block counter(s) will be reset on a read or playback request if:
- the request is non-sequential from a previous read/playback or
- the previous read/playback ended due to a play limit request or
- there have been intervening commands that accessed disk such as record.

In other words, the counter(s) are not reset if a play (or read) is followed by a subsequent read (or play) that is sequential to the previous one and play limit is not used.

To get the total replacement block count, the drives must be idle (i.e., no recording or playback in progress).

If the StreamStor is in bank mode, this command will return the total number of replaced block for the selected bank.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.

Return Value:

The total replacement block count. If no blocks are replaced or an error occurs, zero is returned.

Usage:

```
//
// This example shows FPDP playback using a fill pattern.  After
// playback stops, the total disk replacement block count is retrieved.
//

SHANDLE            xlrDevice;
DWORD              totalRepBlocks;
XLR_RETURN_CODE    xlrStatus=XLR_FAIL;

xlrStatus = XLROpen( 1, &xlrDevice );

// To use fill data, you must first set the skip check dir option.
xlrStatus = XLRSetOption( xlrDevice, SS_OPT_SKIPCHECKDIR );

//
// Establish the fill pattern to use for blocks that cannot be read.
// In this example, the fill pattern is all zeros.
//
xlrStatus = XLRSetFillData( xlrDevice, 0x00000000 );

// Set up to playback over the external port.
xlrStatus = XLRSetMode( xlrDevice,SS_MODE_EXT );

xlrStatus = XLRSetFPDPMode( xlrDevice,SS_FPDP_XMIT,0 );

// Start playback at the beginning of the recording.
xlrStatus = XLRPlayback( xlrDevice, 0, 0 );

     . . . play back data . . .

XLRStop( xlrDevice );

// Get the total number of replaced blocks.
totalRepBlocks = XLRTotalRepBlkCount( xlrDevice );
```

See Also:

XLRDiskRepBlkCount, XLRSetFillData, XLRPlayback, XLRSetOption, XLRSetBankMode and XLRSelectBank.

Structure S_DRIVESTATS

```
typedef struct _S_DRIVESTATS
{
   ULONG range;
   ULONG count;
} S_DRIVESTATS, *PS_DRIVESTATS;
```

Purpose

This structure holds the drive statistics that are collected when the SS_OPT_DRVSTATS option has been set by the XLRSetOption function. Drive statistics are collected as long as the SS_OPT_DRVSTATS option is set.

When SS_OPT_DRVSTATS is set, StreamStor logs the time it takes for each drive to process one block for read or write. Each observation is how long it took to move 0xFFF8 bytes during a record or playback. The time is measured in 15ns (nanoseconds). The observations are counted and accumulated in "bins." The bins are an array of structures of type S_DRIVESTATS.

Each bin holds a count of the number of observations that occurred within that bin's time range. When XLRSetOption is called, the default upper bound for each bin is set and the count for each bin is cleared. The default upper bounds for the bins are:

Bin Number	Range
0	0-75,000
1	75,001-150,000
2	150,001-200,000
3	200,001-250,000
4	250,001-300,000
5	300,001-350,000
6	350,001-400,000
7	400,001 and above

For example, if a write took 3000*15ns, the *count* in bin 0 is incremented by StreamStor. If a write took 220545*15ns, the *count* in bin 3 is incremented. By retrieving the drive statistics for each disk (see XLRGetDriveStats), you may be able to detect if a drive is failing. If the counts on a particular drive are substantially higher than the counts on the other drives, the drive with the high counts is suspect.

Members

range – The upper bound of the range for this bin.

count - The count of the number of observations that fell into the range specified by *range*.

End of Document