NAME

Sys::Virt::Stream − Represent & manage a libvirt stream

DESCRIPTION

The `Sys::Virt::Stream` module represents a stream managed by the virtual machine monitor.

METHODS

my `$st` Sys::Virt::Stream−>new($conn, `$flags`);

Creates a new data stream, ready for use with a stream based API. The optional `$flags` parameter can be used to configure the stream as non-blocking

`$st`−>**abort()**

Abort I/O on the stream. Either this function or `finish` must be called on any stream which has been activated

`$st`−>**finish()**

Complete I/O on the stream. Either this function or `abort` must be called on any stream which has been activated

`$rv` = `$st`−>recv($data, `$nbytes`, `$flags`=0)

Receive up to `$nbytes` worth of data, copying into `$data`. Returns the number of bytes read, or −3 if hole is reached and `$flags` contains RECV_STOP_AT_HOLE, or −2 if I/O would block, or −1 on error. The `$flags` parameter accepts the following flags:

Sys::Virt::Stream::RECV_STOP_AT_HOLE

If this flag is set, the `recv` function will stop reading from stream if it has reached a hole. In that case, −3 is returned and `recv_hole` should be called to get the hole size.

`$rv` = `$st`−>send($data, `$nbytes`)

Send up to `$nbytes` worth of data, copying from `$data`. Returns the number of bytes sent, or −2 if I/O would block, or −1 on error.

`$rv` = `$st`−>recv_hole($flags=0)

Determine the amount of the empty space (in bytes) to be created in a stream's target file when uploading or downloading sparsely populated files. This is the counterpart to `send_hole`. The optional `$flags` parameter is currently unused and defaults to zero if omitted.

`$st`−>send_hole($length, `$flags`=0)

Rather than transmitting empty file space, this method directs the stream target to create `$length` bytes of empty space. This method would be used when uploading or downloading sparsely populated files to avoid the needless copy of empty file space. The optional `$flags` parameter is currently unused and defaults to zero if omitted.

`$st`−>recv_all($handler)

Receive all data available from the stream, invoking `$handler` to process the data. The `$handler` parameter must be a function which expects three arguments, the `$st` stream object, a scalar containing the data received and a data byte count. The function should return the number of bytes processed, or −1 upon error.

`$st`−>send_all($handler)

Send all data produced by `$handler` to the stream. The `$handler` parameter must be a function which expects three arguments, the `$st` stream object, a scalar which must be filled with data and a maximum data byte count desired. The function should return the number of bytes filled, 0 on end of file, or −1 upon error

`$st`−>sparse_recv_all($handler, `$hole_handler`)

Receive all data available from the sparse stream, invoking `$handler` to process the data. The `$handler` parameter must be a function which expects three arguments, the `$st` stream object, a scalar containing the data received and a data byte count. The function should return the number of bytes processed, or −1 upon error. The second argument `$hole_handler` is a function which expects two arguments: the `$st` stream and a scalar, number describing the size of the hole in the stream (in bytes). The `$hole_handler` is expected to return a non-negative number on success

(usually 0) and a negative number (usually −1) otherwise.

`$st`−>sparse_send_all($handler, `$hole_handler`, `$skip_handler`)
Send all data produced by `$handler` to the stream. The `$handler` parameter must be a function which expects three arguments, the `$st` stream object, a scalar which must be filled with data and a maximum data byte count desired. The function should return the number of bytes filled, 0 on end of file, or −1 upon error. The second argument `$hole_handler` is a function expecting just one argument `$st` and returning an array of two elements (`$in_data`, `$section_len`) where `$in_data` has zero or non-zero value if underlying file is in a hole or data section respectively. The `$section_len` then is the number of remaining bytes in the current section in the underlying file. Finally, the third `$skip_handler` is a function expecting two arguments `$st` and `$length` which moves cursor in the underlying file for `$length` bytes. The `$skip_handler` is expected to return a non-negative number on success (usually 0) and a negative number (usually −1) otherwise.

`$st`−>add_callback($events, `$coderef`)
Register a callback to be invoked whenever the stream has one or more events from `$events` mask set. The `$coderef` must be a subroutine that expects 2 parameters, the original `$st` object and the new `$events` mask

`$st`−>update_callback($events)
Change the event mask for a previously registered callback to `$events`

`$st`−>**remove_callback()**;
Remove a previously registered callback

CONSTANTS

Sys::Virt::Stream::NONBLOCK
Create a stream which will not block when performing I/O

Sys::Virt::Stream::EVENT_READABLE
The stream has data available for read without blocking

Sys::Virt::Stream::EVENT_WRITABLE
The stream has ability to write data without blocking

Sys::Virt::Stream::EVENT_ERROR
An error occurred on the stream

Sys::Virt::Stream::EVENT_HANGUP
The remote end of the stream closed

AUTHORS

Daniel P. Berrange <berrange@redhat.com>

COPYRIGHT

LICENSE

SEE ALSO

Sys::Virt, Sys::Virt::Error, `http://libvirt.org`