

**Intel® Media Streaming Library for ST 2110 User Guide**

Intel Corporation, Document Version: 0.1

**LEGAL DISCLAIMER**

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO

LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL

PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S

TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY

WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO

SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT

DESIGNED NOR INTENDED FORANYAPPLICATION IN WHICH THE FAILURE OF THE INTEL

PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting [Intel's Web Site.](http://www.intel.com/)

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

\*Other names and brands may be claimed as the property of others.

Copyright © 2020, Intel Corporation. All Rights reserved.

Table of Contents

[Overview 6](#_Toc51691044)

[Acronyms and Abbreviations 6](#_Toc51691045)

[Requirements 7](#_Toc51691046)

[Operation concept 7](#_Toc51691047)

[Flow diagram 10](#_Toc51691048)

[Key parameters 11](#_Toc51691049)

[API Diagram 12](#_Toc51691050)

[StDevice API group 13](#_Toc51691051)

[StInitLib – deprecated, to be removed after alpha 13](#_Toc51691052)

[StCreateDevice 13](#_Toc51691053)

[StStartDevice 14](#_Toc51691054)

[StDestroyDevice 14](#_Toc51691055)

[StPtpSetClockSource 15](#_Toc51691056)

[StPtpGetClockSource 15](#_Toc51691057)

[StPtpSetParam 16](#_Toc51691058)

[StPtpGetParam 16](#_Toc51691059)

[St21 API group 17](#_Toc51691060)

[St21CreateSession 17](#_Toc51691061)

[St21BindIpAdrr 19](#_Toc51691062)

[St21ListenSession 20](#_Toc51691063)

[St21JoinSession 20](#_Toc51691064)

[St21DropSession 21](#_Toc51691065)

[St21GetFormat 21](#_Toc51691066)

[St21SetParam 22](#_Toc51691067)

[St21GetParam 22](#_Toc51691068)

[St21 Producer API group 24](#_Toc51691069)

[St21RegisterProducer 24](#_Toc51691070)

[St21ProducerStartFrame 26](#_Toc51691071)

[St21ProducerUpdate 26](#_Toc51691072)

[St21ProducerStop 27](#_Toc51691073)

[St21 Consumer API group 27](#_Toc51691074)

[St21RegisterConsumer 27](#_Toc51691075)

[St21ConsumerStartFrame 30](#_Toc51691076)

[St21ConsumerUpdate 30](#_Toc51691077)

[St21ConsumerStop 31](#_Toc51691078)

[Compilation and building 31](#_Toc51691079)

[Installation 31](#_Toc51691080)

[Run 32](#_Toc51691081)

[Sample execution parameters 32](#_Toc51691082)

[Change execution parameters 32](#_Toc51691083)

# Overview

*The Intel® Media Streaming Library for ST 2110 is an easy to implement set of C language APIs that will allow for the development of applications which perform lossless high-speed video transmission over IP networks as specified by the SMPTE 2110 family of standards. SMPTE (Society of Motion Picture and Television Engineers) is a well-established organization that has driven this next generation wide industry adapted set of digital video transmission standards. SMPTE ST 2110 specifies the carriage, synchronization, and description of separate elementary essences for video, audio, and data over IP networks. A wide range of software and hardware products in the video Industry use this standard for real-time production, playout, and other professional media applications.*

*The library will allow a developer to focus on the processing of video frames and lines while utilizing the library to manage the fine details of forming IP packets with RTP encapsulation and decapsulation of efficiently packed pixel data, along with nanosecond precise PTP timing. Up to 100GBps traffic per port of video, audio, and ancillary essences as defined in the ST2110 standard can be generated using this library to develop transmit and receive applications experiencing very low latency (e.g. 1 frame end to end).*

*DPDK (Data Plane Development Kit) is a very popular software solution for the problem of handling high speed transmission of data which bypasses Linux kernel bottlenecks including interrupts and memory copies. A key advantage of the Intel® Media Streaming Library for ST 2110 is that it uses DPDK efficiently and provides high performance processing of packets as they are sent and received from network interface cards.*

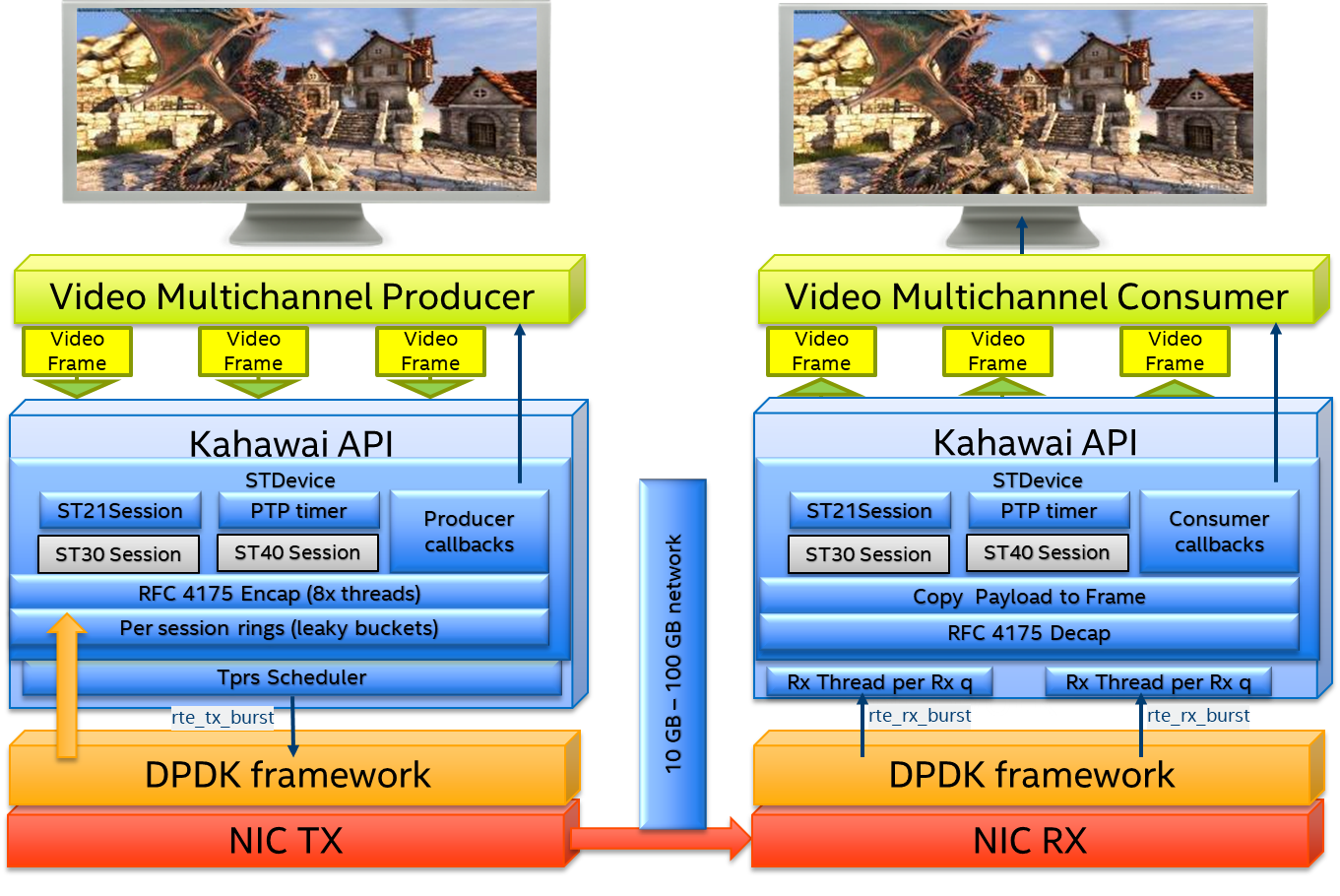
*A wide range of software applications can be developed taking advantage of this library's support for multiple sessions, diverse pixels formats, frame sizes and rates, interlacing, multicasting, and network speeds. In addition to continued optimization and support for even higher network speeds, future versions of this library will allow for containerized applications running in virtual environments, more advanced packet pacing, and supporting stream redundancy for improved transmission reliability.*

## Acronyms and Abbreviations

|  |  |
| --- | --- |
| BDF | Bus:Device:Function – system ID of a PCI device |
| DPDK | Data Plane Development Kit |
| Frame | Refers to video frame |
| Frame Epoch | Time of the frame start equal to the time in which the frame would start under assumption that the 1st frame was send at the UPC epoch (so that the is integer number of complete frames since then) |
| Frame rate | Video frames per second rate |
| KM | Kernel Mode |
| KMD | Kernel Mode Driver |
| MTU | Maximal Transfer Unit of a packet allowed on network, for IP packet on Ethernet this is 1500. |
| Packet | Packet (byte frame) on network |
| PTP | Precision Time Protocol |
| RTP | Real Time Protocol |
| SDP | Session Description Protocol – Sender metadata that helps to identify RTP session format and essence |
| SMPTE | Society of Motion Pictures and Television Engineers |
| St21 Session | RTP Session of Raw Video complying to RFC 4175 |
| St30 Session | RTP Session of PCM Audio |
| St31 Session | RTP Session of AES3 Audio |
| St40 Session | RTP session with Ancillary data |
| UDP | User Datagram Protocol, Layer 4 protocol on the top of IPv4 or IPv6 protocol |
| UM | User Mode |
| UMD | User Mode Driver |
| VRX | Virtual Receive Buffer |
| RVRTP | Raw Video Real Time Protocol |
| UIO | Userspace Input/Output |
| IGB | Intel GigaBit Ethernet Driver |

## Architecture overview

### Sender – receiver transmission stack.



**Video Multichannel Producer Application**

Customer SW that calls the Intel Media Streaming library for ST 2110 and in case of video producer provides callback interface to drain the video buffers,

**Video Multichannel Consumer Application**

In case of video consumer provides callback interface to get notification about received frames or frame slices. The data interface to the library is through the video buffers that contain CbY0CrY1 10bit packet color (5 bytes are used to hold CbY0CrY1 color that are for 2 consecutive pixels).

### StDevice components for Transmitter and Receiver.

**StDevice** implements fundamental library structure in each direction, on which Video, Audio and Ancillary sessions are instantiated. Usually a single device is created for a pair of NIC ethernet ports, so that naturally can provide the redundant path capability – streams addressed for different destinations but having the same timing and payload, so that the same video essence travels a different path in the network. In case of sender device, it initializes the pool of timeslots through dividing the frame time into slices equal to the default packet pacing distance – for example in case of 50fps rate the time slice size is 4.44us for 1080p traffic. Then each slice can send a number of streams, in case of 10Gbps NIC port within 4.44us 4 packets of default video packet size of 1268 can be sent.

**St21Session** component is responsible for raw video session setup and management, for direction of transmitter it registers within device for timeslots that guarantee smooth transition of the video. It is possible the base rate of the device is not compatible with the rate and resolution of the session, please refer to section 6.1.3.1. Each session has callback registered from the sender or receiver application. The session component is responsible for issuing these callbacks to drain the video buffer for the transmitter side and notify about frame completion on the receiver side. Each video session on transmitter side has assigned scheduler ring. The St21Session API is not intended for multi-process communication. The API will work within the same process as library is instanced. To achieve multi-process communication, several approaches are possible, one is to add shared memory to exchange data efficient with other processes and implement inter process notifications on the top of the session callbacks.

**St30Session(not implemented in prototype release)** component is responsible for audio session setup and management (PCM AES67 or AES3), for direction of transmitter it registers within device for time budget within multichannel audio packets that guarantee smooth transition of the audio (which is tiny compared to video). Audio packet per each session is send every 1ms and usually from 2 to 8 channels form a single, bigger audio packet to save the bandwidth. All audio channels are sent via socket API, which then in the library, use one of remaining send device rings (since video cannot occupy more than 90% of the NIC bandwidth).

**St40Session (not implemented in prototype release)**

Ancillary data

**RFC4175 Raw Video RTP** encapsulation component (also called rvrtp rings) performs RTP encapsulation and puts packets to the scheduler ring. It is also responsible for aligning the 1st packet of the frame to the Epoch + Troffset time. The alignment algorithm calculates the time to the next frame beginning and waits in draining the video buffer through missing time slices (Tprs), therefore for example of 50fps frame rate of 1080p session, a step of delay is Trps time of 4.44us, for 25fps is 8.88us.

On the receiver side, **RFC4175 Raw Video RTP** component is responsible for decapsulation of RTP packets (and verification). Each received stream has assigned one of available RX queues, so that multiple threads can easily parallelize reception of the packets of different sessions.

### Flow diagram

## Typical TX application setup for unicast IP destination



\* - not implemented in the prototype release

## Typical TX application setup for multicast IP destination



\* - not implemented in the prototype release

## Typical RX application setup



\* - not implemented in the prototype release

# API Reference



Figure 3. ST Library API Structure diagram

## 2. StDevice API group:

### 2.1 StSetParam

Notation:

st\_status\_t

StGetParam(  
 st\_param\_t prm,  
 st\_param\_val\_t val);

typedef enum st\_param

{

ST21\_FRM\_FIX\_MODE = 10, //configurable, by default enabled for ST21\_FRM\_FIX\_PREV\_N\_2022\_7

ST21\_FRM\_2022\_7\_MODE = 11, //configurable, by default enabled if 2nd port is configured

ST21\_TPRS = 20, //value shall be in nanoseconds, read-only

ST21\_TR\_OFFSET = 21, //value shall be in nanoseconds, read-only

ST21\_FRM\_TIME = 22, //value shall be in nanoseconds, read-only

ST21\_PKT\_TIME = 23, //value shall be in nanoseconds, read-only

ST21\_PIX\_GRP\_SZ = 30, // read-only

ST\_BUILD\_ID = 40, // read-only

ST\_LIB\_VERSION = 41, // read-only

ST\_PTP\_DROP\_TIME = 100,

ST\_PTP\_THRESHOLD = 101,

ST\_SOURCE\_IP = 150,

ST\_DESTINATION\_IP = 151,

ST\_EBU\_TEST = 152,

ST\_SN\_COUNT = 153,

ST\_TX\_ONLY = 154,

ST\_RX\_ONLY = 155,

ST\_MAC = 156,

ST\_OUT\_PORT = 157,

ST\_IN\_PORT = 158,

ST\_FMT\_INDEX = 159,

ST\_DPDK\_PARAMS = 160,

} st\_param\_t;

typedef union

{

uint32\_t valueU32;

uint64\_t valueU64;

char \*strPtr;

} st\_param\_val\_t;

Description:

The function shall be called in order to set library specific parameter like for example: static destination MAC address, enable/disable of printing EBU statistics, etc.

Remarks:

Supported since first SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

### 2.2 StGetParam

Notation:

st\_status\_t

StGetParam(  
 st\_param\_t prm,  
 st\_param\_val\_t \*val);

typedef enum st\_param

{

ST21\_FRM\_FIX\_MODE = 10, //configurable, by default enabled for ST21\_FRM\_FIX\_PREV\_N\_2022\_7

ST21\_FRM\_2022\_7\_MODE = 11, //configurable, by default enabled if 2nd port is configured

ST21\_TPRS = 20, //value shall be in nanoseconds, read-only

ST21\_TR\_OFFSET = 21, //value shall be in nanoseconds, read-only

ST21\_FRM\_TIME = 22, //value shall be in nanoseconds, read-only

ST21\_PKT\_TIME = 23, //value shall be in nanoseconds, read-only

ST21\_PIX\_GRP\_SZ = 30, // read-only

ST\_BUILD\_ID = 40, // read-only

ST\_LIB\_VERSION = 41, // read-only

ST\_PTP\_DROP\_TIME = 100,

ST\_PTP\_THRESHOLD = 101,

ST\_SOURCE\_IP = 150,

ST\_DESTINATION\_IP = 151,

ST\_EBU\_TEST = 152,

ST\_SN\_COUNT = 153,

ST\_TX\_ONLY = 154,

ST\_RX\_ONLY = 155,

ST\_MAC = 156,

ST\_OUT\_PORT = 157,

ST\_IN\_PORT = 158,

ST\_FMT\_INDEX = 159,

ST\_DPDK\_PARAMS = 160,

} st\_param\_t;

typedef union

{

uint32\_t valueU32;

uint64\_t valueU64;

char \*strPtr;

} st\_param\_val\_t;

Description:

The function shall be called in order to retrieve library specific parameter value like for example: static destination MAC address, enable/disable of printing EBU statistics, etc.

Remarks:

Supported since the first SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

### 2.3 StCreateDevice

Notation:

st\_status\_t

StCreateDevice(

st\_device\_t \*inDev, //IN structure with device parameters

const char \*port1Bdf, //IN BDF of primary port PCI device

const char \*port2Bdf, //IN BDF of secondary port PCI device

st\_device\_t \*\*outDev); //OUT created device object w/ fields updated per link capabilities

struct st\_device

{

st\_version\_t ver;//IN requested version of the API

st\_dev\_type\_t type;//IN requested device type, one of producer or consumer

st\_pacer\_type\_t pacerType;//IN requested pacing type

st\_exact\_rate\_t exactRate;//IN requested exact base frame rate

uint32\_t rateGbps;//OUT, determined by a NIC port rate in Gbps, 10-100 are expected values

uint16\_t port[2]; //OUT, DPDK port IDs as selected with the BDFs

uint16\_t mtu;//if > 1500 requested MTU, updated with value possible on the links

uint16\_t maxSt21Sessions;//IN arbitrary maximal number of ST21 video sessions

uint16\_t maxSt31Sessions;//IN arbitrary maximal number of ST21 audio sessions

uint16\_t maxSt40Sessions;//IN arbitrary maximal number of ST40 ancillary sessions

};

Description:

The function is called to initialize the library and create a single device for either a transmit or receive operation. By creating both kind of devices the library is configured to do both operations simultaneously. Only a single device per each direction can be created at a time for a pair of ports.

The device is created on ports (specified by port1Bdf and port2Bdf) that are represented by a PCIe Bus:Device.Function string, for example 0000:af:00.1 - NIC device 0x0 function 0x1 (usually 2nd port) seating on a PCIe bus 0xaf. 2 ports are required to setup a redundant path streaming. i.e. the 2nd specified port acts as a video streaming redundancy mirror (with different IP and VLAN addresses compared to 1st path stream).

The meaning of the other function parameters is as follows:

* inDev structure can be used to force arbitrary number of sessions to a lower value than is available on the NIC port. It also selects the following options for the device:
  + type selects device type either ST\_DEV\_TYPE\_PRODUCER or ST\_DEV\_TYPE\_CONSUMER
  + pacerType selects the sender type,either ST\_2110\_21\_TPW, ST\_2110\_21\_TPNL, or ST\_2110\_21\_TPN
  + exactRate selects the video frame rate either ST\_DEV\_RATE\_29\_97, ST\_DEV\_RATE\_25\_00, ST\_DEV\_RATE\_59\_95, or ST\_DEV\_RATE\_50\_00
  + maxSt21Sessions selects maximal number of ST21 uncompressed video sessions allowed to be created on the device
  + maxSt30Sessions selects maximal number of ST30 uncompressed audio sessions allowed to be created on the device
  + maxSt40Sessions selects maximal number of ST40 ancillary sessions allowed to be created on the device
* outDev pointer is used to return the created device handle that can be used in the following API calls upon successful completion.

Remarks:

Supported since first SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

ST\_OUT\_OF\_MEMORY

### 2.4 StStartDevice

Notation:

st\_status\_t

StStartDevice(st\_device\_t \*dev)

Description:

The function shall be called to start device operation for transmit or receive operation.

Remarks:

Supported since the first SW release.

Return codes and errors:

ST\_OK

ST\_GENERAL\_ERROR

### 2.5 StDestroyDevice

Notation:

st\_status\_t

StDestroyDevice(st\_device\_t \*dev)

Description:

The function is called to free allocated device resources before gracefully closing the application.

Remarks:

Supported since prototype SW release.

Return codes and errors:

ST\_OK

ST\_GENERAL\_ERROR

### 2.6 StPtpSetClockSource

Notation:

st\_status\_t

StPtpSetClockSource(

st\_ptp\_clock\_id\_t \*priClock, //IN pointer to the clock ID of primary PTP Grandmaster

st\_ptp\_clock\_id\_t \*bkpClock);//IN, optionally pointer to the clock ID of backup PTP Grandmaster

Description:

The function is called to enable accepting PTP frames from the specific grandmaster clock source. The primary clock source must be specified, while the backup clock source is optional.

Remarks:

Supported since prototype SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

### 2.7 StPtpGetClockSource

Notation:

st\_status\_t

StPtpGetClockSource(

st\_ptp\_clock\_id\_t \*currClock);//OUT, pointer to the clock ID that is being currently used as Grandmaster

Description:

The function is called to get the active PTP grandmaster ID.

Remarks:

Supported since prototype SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

### 2.8 StPtpSetParam

Notation:

st\_status\_t

StPtpSetParam(

st\_param\_t prm, //IN param ID

uint64\_t val); //IN value to be set

Description:

The function is called to set PTP specific parameters. Supported parameters are:

ST\_PTP\_DROP\_TIME - Drop time after which the backup PTP clock is used instead of primary PTP. Also once the primary clock is back (online?) then it needs to be stable for at least as long as the interval to switch back. In that case val contains threshold (in nanoseconds), the value above which the backup PTP Grandmaster is used as the active Grandmaster.

ST\_PTP\_THRESHOLD - PTP adjustment threshold time above which the Ptp (What is the Ptp timer??) timer gets PTP Grandmaster clock unconditionally as is (since the difference is too large). Otherwise, half of the difference is used to adjust the local PTP clock. If the threhold time is set to 0 then the PTP Grandmaster gets as is (default behavior). Then the value of val is then threshold (in nanoseconds), is the value above which the received value of PTP Grandmaster is used as the local PTP clock s frames from the specific clock source Grandmasters. The primary clock source must be specified while the backup clock source is optional.

Remarks:

Supported since prototype SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

### 2.9 StPtpGetParam

Notation:

st\_status\_t

StPtpSetParam(

st\_param\_t prm, //IN param ID

uint64\_t \*val); //OUT value to be get

Description:

The function is called to retrieve PTP specific parameters. Supported parameters are:

ST\_PTP\_DROP\_TIME - Drop time after which the backup PtP clock is used instead of primary Ptp. Also aonce the primary clock is back then it needs to be stable as least as long this interval to switch back. In that case the location pointed by val is updated with the threshold (in nanoseconds), threshold above which the backup PTP Grandmaster is used as the active Grandmaster.

ST\_PTP\_THRESHOLD - PTP adjustment threshold time above which the Ptp timer gets PTP Grandmaster clock unconditionally as is (since the difference is too large). Otherwise, half of the difference is used to adjust the local PTP clock. In that case the location pointed by val is then updated with the value of threshold (in nanoseconds) that is set in the library.

Remarks:

Supported since prototype SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

# 3. St21 API group

### 3.1 St21CreateSession

Notation:

st\_status\_t

St21CreateSession(

st\_device\_t \*dev, // IN device on which session is created

st21\_session\_t \*inSn, // IN structure of session params

st21\_format\_t \*fmt, // IN session packet's format

st21\_session\_t \*\*outSn) // OUT created session object w/ fields updated respectively

struct st21\_session {

st21\_sn\_flags\_t caps;//IN capabilities of the session (if single or dual path)

uint32\_t ssid; //IN determines dynamic ID of the session

uint16\_t nicPort[2];// derived from device NIC ports, second valid if dual path is supported

uint32\_t timeslot;// assigned timeslot ID [0- 1st timeslot, N] from device timeslots

uint32\_t trOffset;// offset of the timeslot since even EPOCH (calculated by default)

uint32\_t tprs; // time in nanoseconds of 2 consecutive packets begins of the same session

uint32\_t pktTime; // time in nanoseconds of the packet

uint32\_t frameSize;

uint64\_t pktsDrop[ST\_PKT\_DROP\_MAX];

uint64\_t frmsDrop[ST\_FRM\_DROP\_MAX];

uint64\_t pktsLost[ST\_PKT\_LOST\_MAX];

uint64\_t pktsSend;

uint64\_t frmsSend;

uint64\_t pktsRecv;

uint64\_t frmsRecv;

};

struct st21\_format {

st21\_pix\_fmt\_t pixelFmt;

st21\_vscan\_t vscan;

uint32\_t height;

uint32\_t width;

uint32\_t totalLines; // 750 for 720pHD, 1125 for HD, 2250 for UHD

uint32\_t trOffsetLines;// 22 for interlaced HD, 43 for progressive HD

uint32\_t pixelGrpSize;// 3 for RGB, 5 for 422 10 bit - shall match the format - for sanity check

uint32\_t pixelsInGrp; // number of pixels in each pixel group, 2 for 422-10

uint32\_t clockRate; //90k of sampling clock rate

uint32\_t frmRateMul; //60000 or 30000

uint32\_t frmRateDen; //1001 or 1000

st21\_pkt\_fmt\_t pktFmt;// if single, dual or more lines of RFC4175 or other format

uint32\_t pixelsInPkt; // number of pixels in each line

uint32\_t pktsInLine; // number of packet per each line

uint32\_t pktSize; //pkt size w/o VLAN header

uint32\_t frameTime; // time in nanoseconds of the frame

uint32\_t pktsInFrame; //packets in frame

};

Description:

The function is called to create a ST2110-21 RTP raw video session (complying to RFC 4175) on a transmit of the receive device (the device direction determines the direction of the created session). The video type is determined by the format structure, i.e. interlaced or progressive scanning, resolution and frame rate. The created session frame rate shall be compatible with the device setting according to the table below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| device rate (exactRate) | | 25 | 50 | 29 | 59 |
| session rate progressive | 25 | Y | N | N | N |
| session rate progressive | 50 | Y | Y | N | N |
| session rate progressive | 29 | N | N | Y | N |
| session rate progressive | 59 | N | N | Y | Y |
| session rate interlaced | 25 | Y | N | N | N |
| session rate interlaced | 50 | Y | N | N | N |
| session rate interlaced | 29 | N | N | Y | N |
| session rate interlaced | 59 | N | N | Y | N |

The meaning of the other function parameters is as follows:

* inSn structure is to can be used to set session capabilities and ssid.
* outSn pointer shall be used to return the created session handle that is to be used in the following API calls upon successful completion.

Remarks:

Supported since prototype SW release.

Return codes and errors:

ST\_OK

ST\_SN\_ERR\_NO\_TIMESLOT

ST\_SN\_ERR\_RATE\_NO\_FIT

ST\_FMT\_ERR\_BAD\_PIX\_FMT : ST\_FMT\_ERR\_LAST

### 3.2 St21BindIpAdrr

Notation:

st\_status\_t

St21BindIpAddr(

st21\_session\_t \*sn, // IN referred session

st\_addr\_t \*addr, // IN addr structure (union for IPV4 and IPV6)

uint16\_t nicPort); // IN nic port on which that address is valid (one of session NIC ports)

typedef enum st\_addr\_opt {

ST\_ADDR\_UCAST\_IPV4 = 0x1,

ST\_ADDR\_MCAST\_IPV4 = 0x2,

ST\_ADDR\_UCAST\_IPV6 = 0x4,

ST\_ADDR\_MCAST\_IPV6 = 0x8,

ST\_ADDR\_VLAN\_TAG = 0x10,

ST\_ADDR\_VLAN\_DEI = 0x20,

ST\_ADDR\_VLAN\_PCP = 0x40,

ST\_ADDR\_IP\_ECN = 0x100,

ST\_ADDR\_IP\_DSCP = 0x200,

} st\_addr\_opt\_t;

/\*\*

\* Structure for connection address of IPv4/v6 and UDP ports

\*/

struct st\_addr {

st\_addr\_opt\_t options; // IN options selecting union fields to be used

union {

struct sockaddr\_in addr4;

struct sockaddr\_in6 addr6;

} src;

union {

struct sockaddr\_in addr4;

struct sockaddr\_in6 addr6;

} dst;

union {

struct {

uint16\_t tag : 12; //IN VLAN tag if

uint16\_t dei : 1;

uint16\_t pcp : 3; // IN referred session

};

uint16\_t vlan;

};

union {

struct {

uint8\_t ecn : 2;

uint8\_t dscp : 6;

};

uint8\_t tos;

};

};

Description:

The function is called to set destination and source IP addresses, VLAN tag, and DiffServ priority of the streams through IP dscp or VLAN pcp. The remaining fields are expected to be 0.

This function in case of setting unicast IP address on a session belonging to a sender device is expected to send ARP request on the specific port. In case of a session created on a receiver device by setting Multicast IP address as destination address, IGMP message of a Add Member to the group selected with that address is send, so that switches listening to IGMP with IGMP Snooping can learn this IP Multicast group on the receiver switch port.

Remarks:

Supported since prototype SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

### St21ListenSession

Notation:

st\_status\_t

St21ListenSession(

st21\_session\_t \*sn, // IN referred session

st\_addr\_t \*addr); // IN addr structure (union for IPV4 and IPV6

Description:

The function is called formulticast addresses only (ST\_ADDR\_MCAST\_IPV4 shall be specified in options field of the st\_addr\_t structure) on the producer side to initiate sending IGMP Queries of the multicast group so that the receivers can respond to these queries with IGMP Member Report packets. The IGMP group address is extracted from the dst field of the st\_addr\_t structure while the other structure members are ignored.

Remarks:

Supported since Beta SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

### St21JoinSession

Notation:

st\_status\_t

St21JoinSession(

st21\_session\_t \*sn, // IN referred session

st\_addr\_t \*addr); // IN addr structure (union for IPV4 and IPV6)

Description:

The function is called for the multicast addresses only (ST\_ADDR\_MCAST\_IPV4 is specified in options field of the st\_addr\_t structure) on the consumer side to initiate responding to IGMP Queries of the multicast group with IGMP Member Report packets. The IGMP group address is extracted from the dst field of the st\_addr\_t structure while the other structure members are ignored. Thanks to these responses, the network switch implementing IGMP Snooping can learn the IGMP MAC addresses of the groups to which there are active multicast receivers. This allows the multicast packets to be passed to these enabled switch ports.

Remarks:

Supported since Beta SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

### St21DropSession

Notation:

st\_status\_t

St21DropSession(

st21\_session\_t \*sn); // IN referred session

Description:

The function is called for stopping IGMP activities, both on the producer (transmitter) side that is sending IGMP Queries of the multicast group and on the consumer (receiver) side which is responding to the IGMP queries with IGMP Member Report packets.

Remarks:

Supported since Beta SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

### St21GetFormat

Notation:

st\_status\_t

St21GetFormat(

st21\_session\_t \*sn, // IN referred session

st21\_format\_t \*fmt); // OUT format

Description:

The function is called to retrieve the format information of the existing session.

Remarks:

Supported since prototype SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

### St21SetParam

Notation:

st\_status\_t

St21SetParam(

st21\_session\_t \*sn, // IN referred session

st\_param\_t prm, // IN parameter ID

uint32\_t val); // IN value

Description:

The function is called setting session specific parameters for example: enable/disable redundancy path, enable/disable slice level notifications, etc.

Remarks:

Supported since Beta SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

### St21GetParam

Notation:

st\_status\_t

St21SetParam(

st21\_session\_t \*sn, // IN referred session

st\_param\_t prm, // IN parameter ID

uint32\_t \*val); // OUT value

Description:

The function is called for getting session specific parameters for example: state of enable/disable of the redundancy path, state of enable/disable of the slice level notifications, etc.

Remarks:

Supported since Beta SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

## 

## St21 Producer API group

### St21RegisterProducer

Notation:

st\_status\_t

St21RegisterProducer (

st21\_session\_t \*sn, // IN session pointer

st21\_producer\_t \*prod)// IN producer structure to register

struct st21\_producer

{

void \*appHandle;

st21\_prod\_type\_t prodType; // slice mode or frame mode w/ or w/o timestamp

uint32\_t frameSize;

uint32\_t sliceSize;

uint32\_t sliceCount; // count of slices in Frame if slice mode

St21GetNextFrameBuf\_f St21GetNextFrameBuf;

St21GetNextSliceOffset\_f St21GetNextSliceOffset;

St21GetFrameTmstamp\_f St21GetFrameTmstamp;

St21NotifyFrameDone\_f St21NotifyFrameDone;

St21NotifySliceDone\_f St21NotifySliceDone;

St21NotifyStopDone\_f St21NotifyStopDone;

St21BuildUdpPkt\_f St21BuildUdpPkt;

};

typedef struct st21\_producer st21\_producer\_t;

typedef enum st21\_prod\_type {

ST21\_PROD\_INVALID,

ST21\_PROD\_P\_FRAME,//producer of complete frames in progressive mode

ST21\_PROD\_P\_FRAME\_SLICE,//producer of slices of progressive frames

ST21\_PROD\_P\_FRAME\_TMSTAMP,//producer of complete frames and having SDI timestamp already determined

ST21\_PROD\_P\_SLICE\_TMSTAMP,//producer of sliced frames and having SDI timestamp already determined at the frame level

ST21\_PROD\_I\_FIELD,//producer of interlaced fields

ST21\_PROD\_I\_FIELD\_SLICE,//producer of slices of interlaced fields

ST21\_PROD\_I\_FIELD\_TMSTAMP,//producer of interlaced fields and having SDI timestamp already determined

ST21\_PROD\_I\_SLICE\_TMSTAMP,//producer of slices of interlaced fields and having SDI timestamp already determined at the field level

ST21\_PROD\_RAW\_UDP,//producer that assembles own UDP frames, it uses only callback of St21BuildUdpPkt\_f

} st21\_prod\_type\_t;

Description:

The function is called to register a live video producer for video streaming within the ST2110-21 RTP raw video session. The producer should implement callbacks depending on its type as described by the below table. Notice that for ST21\_PROD\_RAW\_UDP only callbacks of St21BuildUdpPkt\_f should implemented.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function | ST21\_PROD\_P\_FRAME  ST21\_PROD\_I\_FIELD | ST21\_PROD\_P\_FRAME\_SLICE  ST21\_PROD\_I\_FIELD\_SLICE | ST21\_PROD\_P\_FRAME\_TMSTAMP  ST21\_PROD\_I\_FIELD\_TMSTAMP | ST21\_PROD\_P\_FRAME\_SLICE\_TMSTAMP  ST21\_PROD\_I\_FIELD\_SLICE\_TMSTAMP |
| St21GetNextFrameBuf\_f | Required | Required | Required | Required |
| St21GetNextSliceOffset\_f | n/a | Required | n/a | Required |
| St21GetFrameTmstamp\_f | n/a | n/a | n/a | Required |
| St21NotifyFrameDone\_f | Required | Required | Required | Required |
| St21NotifySliceDone\_f | n/a | Required | n/a | Required |
| St21NotifyStopDone\_f | Required | Required | Required | Required |

The meanings of the producer types are the following:

|  |  |  |
| --- | --- | --- |
| Type | Description | Support since |
| ST21\_PROD\_P\_FRAME | Basic producer w/ the frame buffer granularity level of complete frames in progressive mode | Prototype |
| ST21\_PROD\_I\_FIELD | Basic producer w/ the interlaced field buffer granularity level of complete fields in interlaced mode | Prototype |
| ST21\_PROD\_P\_FRAME\_SLICE | Producer as the frame buffer granularity level of slices of the frames in progressive mode. Slices must be complete lines and continuous fragments of the frame and their practical number shall be limited due to performance reasons to not more than several dozens. | Beta |
| ST21\_PROD\_I\_FIELD\_SLICE | Producer as the interlaced field granularity level of complete fields in interlaced mode. Slices must be complete lines and continuous fragments of the field and their practical number shall be limited due to performance reasons to not more than several dozens. | Beta |
| ST21\_PROD\_P\_FRAME\_TMSTAMP | Basic producer w/ the frame buffer granularity level of complete frames in progressive mode that has timestamp determined by the producer externally to the library | Beta |
| ST21\_PROD\_I\_FIELD\_TMSTAMP | Basic producer w/ the interlaced field buffer granularity level of complete fields in interlaced mode that has timestamp determined by the producer externally to the library | Beta |
| ST21\_PROD\_P\_SLICE\_TMSTAMP | Producer as the frame buffer granularity level of slices of the frames in progressive mode. Slices must be complete lines and continuous fragments of the frame and their practical number shall be limited due to performance reasons to not more than several dozens. | Beta |
| ST21\_PROD\_I\_SLICE\_TMSTAMP | Producer as the interlaced field granularity level of complete fields in interlaced mode. Slices must be complete lines and continuous fragments of the field and their practical number shall be limited due to performance reasons to not more than several dozens. | Beta |
| ST21\_PROD\_RAW\_UDP | Producer that assembles by their own UDP frames, it uses only callback of St21BuildUdpPkt\_f | Beta |

Other parameters have the following meaning:

* appHandle handle passed back to the producer application upon each callback
* frameSize size of the frame shall match the session setting
* sliceSize size of the slice for the producer (determines session settings)
* sliceCount number of slices for the producer (determines session settings)

Remarks:

Supported since prototype SW release with basic frame/field granularity.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

ST\_SN\_ERR\_NO\_TIMESLOT

ST\_BAD\_PRODUCER

ST\_PROD\_ALREADY\_REGISTERED

### St21ProducerStartFrame

Notation:

st\_status\_t

St21ProducerStartFrame (

st21\_session\_t \*sn, // IN referred session

uint8\_t \*frameBuf, //IN 1st frame buffer for the session

uint32\_t linesOffset,//IN offset in complete lines of the frameBuf to which producer filled the buffer

uint32\_t tmstamp, //IN if not 0 then 90kHz timestamp of the frame

uint64\_t ptpTime); //IN if not 0 start new frame at the given PTP timestamp

Description:

The function is called for start streaming of the producer session. If both tmstamp and ptpTime are specified then the producer session starts at those selected moments (which might be in the future as specified by ptpTime or within the current frame epoch if ptpTime to epoch difference does not exceeds TRoffset specified for that session.

Remarks:

Supported since Beta SW release.

Return codes and error:

ST\_OK

ST\_INVALID\_PARAM

### St21ProducerUpdate

Notation:

st\_status\_t

St21ProducerUpdate(

st21\_session\_t \*sn, // IN referred session

uint8\_t \*frameBuf, //IN 1st frame buffer for the session

uint32\_t linesOffset)//IN offset in complete lines of the frameBuf to which producer filled the buffer

Description:

The function is called for restartting video streaming of the producer session after the session was on hold due to a slow producer.

Remarks:

Supported since Beta SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

### St21ProducerStop

Notation:

st\_status\_t

St21ProducerStop (

st21\_session\_t \*sn) // IN referred session

Description:

The function is called for stopping video streaming of the producer session.

Remarks:

Supported since Beta SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

## 4. St21 Consumer API group

### St21RegisterConsumer

Notation:

st\_status\_t

St21RegisterConsumer (

st21\_session\_t \*sn, // IN session pointer

st21\_consumer\_t \*cons)// IN consumer structure to register

struct st21\_producer

{

void \*appHandle;

st21\_cons\_type\_t consType; // slice mode or frame mode w/ or w/o timestamp

uint32\_t frameSize;

uint32\_t sliceSize;

uint32\_t sliceCount; // count of slices in Frame if slice mode

St21GetNextFrameBuf\_f St21GetNextFrameBuf;

St21GetNextSliceOffset\_f St21GetNextSliceOffset;

St21NotifyFrameRecv\_f St21NotifyFrameRecv;

St21NotifyFrameSlice\_f St21NotifySliceRecv;

St21PutFrameTmstamp\_f St21PutFrameTmstamp;

St21NotifyFrameDone\_f St21NotifyFrameDone;

St21NotifySliceDone\_f St21NotifySliceDone;

St21NotifyStopDone\_f St21NotifyStopDone;

St21RecvUdpPkt\_f St21RecvUdpPkt;

};

typedef struct st21\_consumer st21\_consumer\_t;

typedef enum st21\_cons\_type {

ST21\_CONS\_INVALID,

ST21\_CONS\_P\_FRAME,//consumer of complete frames in progressive mode

ST21\_CONS\_P\_FRAME\_SLICE,//consumer of slices of progressive frames

ST21\_CONS\_P\_FRAME\_TMSTAMP,//consumer of complete frames and reusing RTP timestamp as SDI timestamp

ST21\_CONS\_P\_SLICE\_TMSTAMP,//consumer of sliced frames and reusing RTP timestamp as SDI timestamp

ST21\_CONS\_I\_FIELD,//consumer of interlaced fields

ST21\_CONS\_I\_FIELD\_SLICE,//consumer of slices of interlaced fields

ST21\_CONS\_I\_FIELD\_TMSTAMP,//consumer of interlaced fields and reusing RTP timestamp as SDI timestamp

ST21\_CONS\_I\_SLICE\_TMSTAMP,//consumer of slices of interlaced fields and reusing RTP timestamp as SDI timestamp

ST21\_CONS\_RAW\_UDP,//consumer that assembles own UDP frames, it uses only callback of St21RecvUdpPkt\_f

} st21\_cons\_type\_t;

Description:

The function is called to register a live video consumer for video streaming within the ST2110-21 RTP raw video session. The consumer should implement callbacks depending on its type as described by the below table. Notice that for ST21\_CONS\_RAW\_UDP only callbacks of St21RecvUdpPkt\_f shall be implemented.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function | ST21\_CONS\_P\_FRAME  ST21\_CONS\_I\_FIELD | ST21\_CONS\_P\_FRAME\_SLICE  ST21\_CONS\_I\_FIELD\_SLICE | ST21\_CONS\_P\_FRAME\_TMSTAMP  ST21\_CONS\_I\_FIELD\_TMSTAMP | ST21\_CONS\_P\_FRAME\_SLICE\_TMSTAMP  ST21\_CONS\_I\_FIELD\_SLICE\_TMSTAMP |
| St21GetNextFrameBuf\_f | Required | Required | Required | Required |
| St21GetNextSliceOffset\_f | n/a | Required | n/a | Required |
| St21NotifyFrameRecv\_f | Required | Required | Required | Required |
| St21NotifySliceRecv\_f | n/a | Required | n/a | Required |
| St21PutFrameTmstamp\_f | n/a | n/a | n/a | Required |
| St21NotifyFrameDone\_f | Required | Required | Required | Required |
| St21NotifySliceDone\_f | n/a | Required | n/a | Required |
| St21NotifyStopDone\_f | Required | Required | Required | Required |

The meanings of the producer types are the following:

|  |  |  |
| --- | --- | --- |
| Type | Description | Support since |
| ST21\_CONS\_P\_FRAME | Basic consumer w/ the frame buffer granularity level of complete frames in progressive mode | Prototype |
| ST21\_CONS\_I\_FIELD | Basic consumer w/ the interlaced field buffer granularity level of complete fields in interlaced mode | Prototype |
| ST21\_CONS\_P\_FRAME\_SLICE | Consumer as the frame buffer granularity level of slices of the frames in progressive mode. Slices must be complete lines and continuous fragments of the frame and their practical number shall be limited due to performance reasons to not more than several dozens. | Beta |
| ST21\_CONS\_I\_FIELD\_SLICE | Consumer as the interlaced field granularity level of complete fields in interlaced mode. Slices must be complete lines and continuous fragments of the field and their practical number shall be limited due to performance reasons to not more than several dozens. | Beta |
| ST21\_CONS\_P\_FRAME\_TMSTAMP | Basic consumer w/ the frame buffer granularity level of complete frames in progressive mode that has timestamp determined by the consumer externally to the library | Beta |
| ST21\_CONS\_I\_FIELD\_TMSTAMP | Basic consumer w/ the interlaced field buffer granularity level of complete fields in interlaced mode that has timestamp determined by the consumer externally to the library | Beta |
| ST21\_CONS\_P\_SLICE\_TMSTAMP | Consumer as the frame buffer granularity level of slices of the frames in progressive mode. Slices must be complete lines and continuous fragments of the frame and their practical number shall be limited due to performance reasons to not more than several dozens. | Beta |
| ST21\_CONS\_I\_SLICE\_TMSTAMP | Consumer as the interlaced field granularity level of complete fields in interlaced mode. Slices must be complete lines and continuous fragments of the field and their practical number shall be limited due to performance reasons to not more than several dozens. | Beta |
| ST21\_CONS\_RAW\_UDP | Consumer that assembles by their own UDP frames, it uses only callback of St21RecvUdpPkt\_f | Beta |

Other parameters have the following meaning:

* appHandle handle passed back to the consumer application upon each callback
* frameSize size of the frame shall match the session setting
* sliceSize size of the slice for the consumer (determines session settings)
* sliceCount number of slices for the consumer (determines session settings)

Remarks:

Supported since Prototype SW release with basic frame/field granularity.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

ST\_SN\_ERR\_NO\_TIMESLOT

ST\_BAD\_CONSUMER

ST\_CONS\_ALREADY\_REGISTERED

### St21ConsumerStartFrame

Notation:

st\_status\_t

St21ConsumerStartFrame (

st21\_session\_t \*sn, // IN referred session

uint8\_t \*frameBuf, //IN 1st frame buffer for the session

uint32\_t linesOffset,//IN offset in complete lines of the frameBuf to which consumer filled the buffer

uint32\_t tmstamp, //IN if not 0 then 90kHz timestamp of the frame

uint64\_t ptpTime); //IN if not 0 start new frame at the given PTP timestamp

Description:

The function is called to start receivingthe consumer session. If both of tmstamp and ptpTime are specified then the consumer session starts at those selected moments (which might be in the future as specified by ptpTime or within the current frame epoch if ptpTime to epoch difference does not exceeds TRoffset specified for that session.

Remarks:

Supported since Beta SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

### St21ConsumerUpdate

Notation:

st\_status\_t

St21ConsumerUpdate(

st21\_session\_t \*sn, // IN referred session

uint8\_t \*frameBuf, //IN 1st frame buffer for the session

uint32\_t linesOffset)//IN offset in complete lines of the frameBuf to which consumer can fill the buffer

Description:

The function is called to restart video streaming the consumer session after the session was on hold due to a slow consumer.

Remarks:

Supported since Beta SW release.

Return codes and errors:

ST\_OK

ST\_INVALID\_PARAM

### St21ConsumerStop

Notation:

st\_status\_t

St21ConsumerStop (

st21\_session\_t \*sn) // IN referred session

Description:

The function is called to stop video streaming of the consumer session.

Remarks:

Supported since Beta SW release.

Return codes and errors (some explained):

ST\_OK

ST\_INVALID\_PARAM

# Using Software

**Recommended OS - Ubuntu 18.04 LTS with kernel 5.3**

1. Edit /etc/default/grub file

vim /etc/default/grub

2. Change following line

GRUB\_CMDLINE\_LINUX\_DEFAULT="splash default\_hugepagesz=1G hugepagesz=1G hugepages=8 nomodeset"

3. Edit /etc/fstab file

vim /etc/fstab

4. Add following line

nodev /mnt/huge\_1GB hugetlbfs pagesize=1GB 0 0

5. Update your software:

Sudo apt update

Sudo apt upgrade

6. Install software:

sudo add-apt-repository "deb http://us.archive.ubuntu.com/ubuntu/ saucy universe multiverse"

sudo add-apt-repository "deb http://us.archive.ubuntu.com/ubuntu/ saucy-updates universe multiverse"

apt install -y dpdk-igb-uio-dkms gcc cmake python python3 python3-pip libnuma-dev pkg-config libsdl2\*-dev

pip3 install meson ninja

7. Enable UIO driver:

modprobe uio

8. Configure DPDK:

8.1 Preparation of sources

8.1.1 Using latest stable version of DPDK library from GitHub (preferred)

Clone DPDK repo

git clone https://github.com/DPDK/dpdk.git <NameOfBaseDir>

Checkout to v20.08 version

cd <NameOfBaseDir>

git checkout --no-track -b v20.08 v20.08

8.1.2 Using internal copy on Intel GitLab (not use it)

git clone -b dpdk ssh://git@gitlab.devtools.intel.com:29418/VEI/dpdk\_media\_streamer.git <NameOfBaseDir>

pushd <NameOfBaseDir>

git remote add dpdk.org <https://dpdk.org/git/dpdk>

git remote add github.com <https://github.com/DPDK/dpdk.git>

8.2 Configure library features

8.2.1 Set meson option in meson\_options.txt

enable\_kmods => true use\_hpet => true

8.2.2 Add new build options in /meson\_options.txt

option('support\_ieee1588', type: 'boolean', value: true, description: 'support IEEE1588(PTP) ')

8.2.3 Add configuration to /config/meson.build around line 230 (preferred)

dpdk\_conf.set('RTE\_LIBRTE\_IEEE1588', get\_option('support\_ieee1588'))

8.3 Build and install

meson <NameOfOutDir>

cd <NameOfOutDir>

ninja

sudo ninja install

sudo ldconfig

pkg-config --cflags libdpdk

pkg-config --libs libdpdk

cd ..

8.4 Remarks

If you want to uninstall DPDK library safely use command:

cd <NameOfOutDir>

ninja uninstall

# Build and run

### Build with installation

1. Get resources

Clone Intel® Media Streaming Library for ST 2110 repo

git clone <https://gitlab.devtools.intel.com/VEI/dpdk_media_streamer.git> cd dpdk\_media\_streamer

2. Build library

cd <MediaStreamerDir>

meson <NameOfLibOutDir>

cd <NameOfLibOutDir>

ninja

ninja install

3. Build applications

cd <MediaStreamerDir>/app

meson <NameOfAppOutDir>

cd <NameOfAppOutDir>

ninja

### Build without installation

1. Get resources

Clone Intel® Media Streaming Library for ST 2110 repo

git clone <https://gitlab.devtools.intel.com/VEI/dpdk_media_streamer.git> cd dpdk\_media\_streamer

2. Build library

cd <MediaStreamerDir>

meson <NameOfLibOutDir>

cd <NameOfLibOutDir>

ninja

3. Build applications

cd <MediaStreamerDir>/app

meson <NameOfAppOutDir> -Duselocalmslib=true

cd <NameOfAppOutDir>

ninja

4. Run

${NameOfAppOutDir}/rx\_app/RxApp --in\_port <input PCI device address> --mac <destination MAC address> --ip <destination IP address> --sip <source IP address> --o\_port <output PCI device address> --

Available parameters:

-h : print help info (optional)

-v : print versions info (optional)

--mac <MAC addr> : destination MAC address (required)

--ip <IP addr> : destination IP address (required)

--sip <IP addr> : source IP address (required)

--mip : enable multicast flag

--ebu : enable EBU compatibility logs

--port <UDP port> : base port from which to iterate sessions port IDs (optional)

--rx : run receive mode only (optional)

--tx : run transmit mode only (optional)

--rgba : input / output buffers are in rgba format

--yuv10be : input / output buffers are in yuv10be format

--format <fmt string> : select frame format e.g. a1080i50 = aya 1080 interlaced 50fps (optional)

: e.g. i720p29 = intel 720 progressive 29.97fps

: e.g. i1080p59 = intel 1080 progressive 59.94fps

: e.g. i2160p59 = intel 2160 progressive 59.94fps

: e.g. i1080i29 = intel 1080 interlaced 29.97fps

: e.g. a1080p59 = aya 1080 progressive 59.94fps

--s\_count <number of sessions> : number of sessions (optional)

--o\_port <PCI device address> : output interface PCI device address (required)

--in\_port <PCI device address> : input interface PCI device address (required)

--ptp <hhhhhh.hhhh.hhhhhh> : master clock id

--log\_level <user,level<info/debug/error>> : enable additional logs

4. Running application using environment variables

Edit /etc/environment file

vim /etc/environment

Set following variables

RTE\_SDK=/home/dpdk

RTE\_TARGET=x86\_64-native-linux-gcc

IN\_PORT=0000:af:00.1 # PCI addres of NIC to recv frames

OUT\_PORT=0000:af:00.0 # PCI adress of NIC to send frames

OUT\_IP=192.168.0.2 # IP addres for sender

DEST\_IP=192.168.0.1 # IP addres of destination

DEST\_MAC=12:34:56:ab:cd:ef # MAC of destination

ST\_NIC\_RATE\_GBPS=ST\_NIC\_RATE\_SPEED\_10GBPS

ST\_RX='--rx'

${NameOfAppOutDir}/rx\_app/RxApp ${ST\_RX-} --in\_port $IN\_PORT --mac $DEST\_MAC --ip $DEST\_IP --sip $OUT\_IP --o\_port $OUT\_PORT --

# Appendix A

Error codes

|  |  |
| --- | --- |
| ST\_OK | upon successful completion |
| ST\_INVALID\_PARAM | upon using invalid value in one of inDev structure parameters |
| ST\_OUT\_OF\_MEMORY | generated if the report buffer cannot be created due to not enough memory available, and the location pointed by queryHandle returns NULL |
| ST\_GENERAL\_ERROR | upon a critical system condition (indicates memory corruption) |
| ST\_SN\_ERR\_NO\_TIMESLOT | upon lack of available timeslots for the sessions on device |
| ST\_SN\_ERR\_RATE\_NO\_FIT | upon incompatibility of frame rate of the created session and the one set for the device |
| ST\_FMT\_ERR\_BAD\_PIX\_FMT : ST\_FMT\_ERR\_LAST | error from this range indicates specific format error |
| ST\_BAD\_PRODUCER | upon incompatibility of frame size or slice size / count |
| ST\_BAD\_CONSUMER | upon incompatibility of frame size or slice size / count |
| ST\_CONS\_ALREADY\_REGISTERED | if the consumer is already present on the session |
| ST\_PROD\_ALREADY\_REGISTERED | if the producer is already present on the session |