

# Sangoma Media Transcoding

## API v1.1.4

November 2, 2010

Document Version 3



## Table Of Contents

SNGTC Media Transcoding Solutions .....	3
SNGTC Media Transcoding Architecture .....	4
SNGTC libsng-tc Library API .....	5
SNGTC Media Transcoding Distributed Architecture .....	6
SNGTC libsngtc-node Library API .....	7
SNGTC Media Transcoding Files .....	9
Binary Libraries .....	9
libsng-tc: header files .....	9
libsngtc-node: header files .....	9
SNGTC Media Transcoding Functions - List .....	10
SNGTC Media Transcoding Functions .....	11
sngtc_detect_init_modules .....	11
sngtc_activate_modules .....	12
sngtc_deactivate_modules .....	12
sngtc_create_transcoding_session .....	13
sngtc_free_transcoding_session .....	14
SNGTC Media Transcoding Structures .....	15
sngtc_init_cfg .....	15
sngtc_codec_request .....	16
sngtc_codec_request_leg .....	16
sngtc_codec_reply .....	17
sngtc_codec_reply_leg .....	18
SNGTC Sample Code Library .....	19
Library Initialization .....	19
Library Operation .....	22
Library Callbacks .....	23
Library Shutdown .....	24
APPENDIX .....	25
SNGTC Create RTP Session – New .....	25
SNGTC Create RTP Session – Cached .....	27
SNGTC Free RTP Session – Cached .....	28
SNGTC Free RTP Session – New .....	29

## SNGTC Media Transcoding Solutions

- Sangoma D-Series of Transcoding Cards
- Ethernet Drivers
  - PCIe – Broadcom
  - PCI – Micrel
- Sangoma Transcoding API Library
- Codec applications on Linux
  - Codec module for Asterisk
  - Codec module for FreeSWITCH

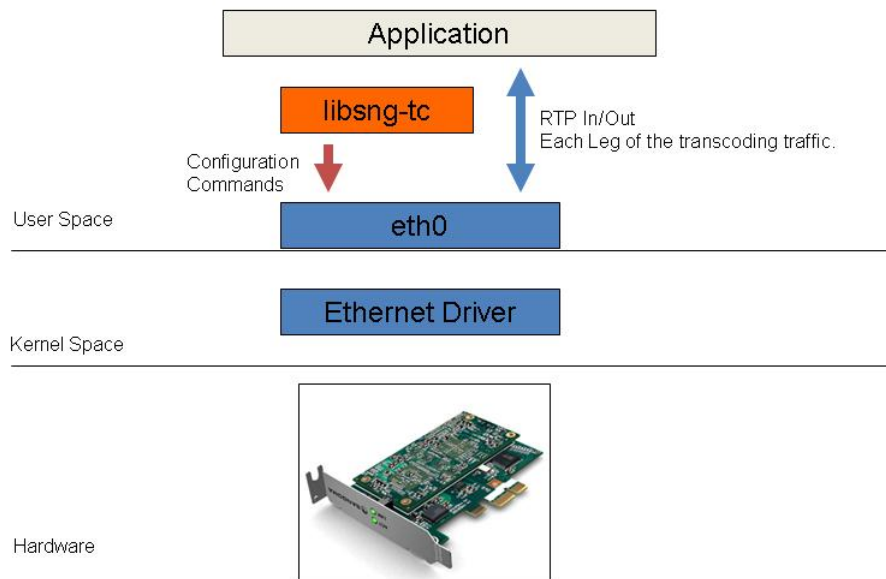


## SNGTC Media Transcoding Architecture

- The libsng-tc library provides API access to Sangoma D-Series transcoding cards.
- All communication between libsng-tc and the hardware is Ethernet based.
- Media transcoding is carried over RTP to and from the card.
- All libsng-tc functions are thread safe.
- An example of user space application would be Asterisk or FreeSWITCH or any custom application.
- Supported for Linux & Windows

11/1/2010

## Media Transcoding Architecture



## SNGTC libsng-tc Library API

The libsng-tc library is provided in binary format on Linux, Windows for both 32bit and 64 bit architectures.

The main functions of libsng-tc are:

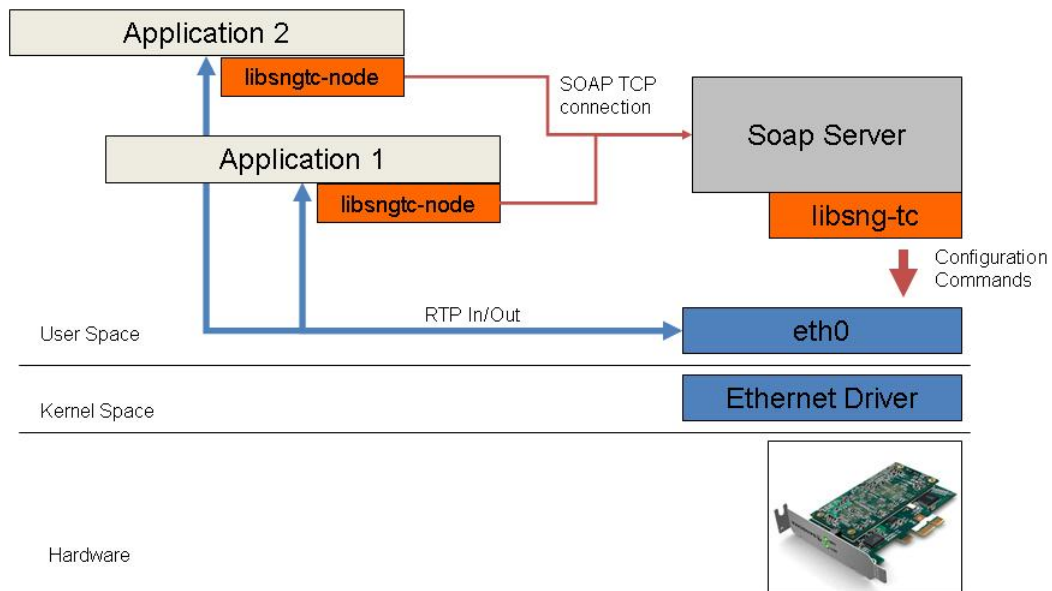
- **Auto detect Sangoma transcoding modules**  
During this step all user defined local Ethernet interfaces are probed for Sangoma transcoding modules.
- **Initialize detected modules for use.**  
All detected Sangoma transcoding modules on all LAN networks are reset and initialized. The firmware and low level configuration is checked.
- **Group all detected modules as unified resource.**  
All detected Sangoma transcoding modules are grouped into a resource pool so that user application does not have to know where each module is located.
- **Create transcoding resource**  
When a resource is requested by the application, the library configures the Sangoma transcoding hardware for an RTP Session and codec information that relates to each leg of the call. After this point user application can start transmitting and receiving RTP voice to and from the transcoding hardware.
- **Free transcoding resource**  
Sangoma transcoding hardware RTP session is stopped and freed.

## SNGTC Media Transcoding Distributed Architecture

- High Level library: libsngtc-node
- Identical API as libsng-tc.
- All API calls are passed via SOAP protocol to the SOAP Server where commands are executed using libsng-tc library.
- Multiple applications can use a single hardware resource simultaneously.
- Applications can access Sangoma transcoding resource from the internet.
- All libsngtc-node functions are thread safe.
- An example of user space application would be Asterisk or FreeSWITCH or any custom application.

11/1/2010

## Media Transcoding Distributed Architecture



## SNGTC libsngtc-node Library API

The libsngtc-node library works as part of the distributed setup in conjunction with a Sangoma SOAP Server. The SOAP Server must be running before on tries to use the libsngtc-node calls.

The SOAP Server is nothing but a SOAP to libsng-tc converter. This architecture is very convenient since applications written for libsng-tc and libsngtc-node are virtually identical.

The only difference between libsng-tc and libsngtc-node applications are the header files, as indicated on "SNGTC Media Transacting Files" section.

Since the SOAP server starts before any libsngtc-node application, it must execute detect and activate commands. Thus detect and activate commands in libsngtc-node are dummy and do not really do anything except keep API compatibility between two APIs.

Note: that this will change in future release: the libsngtc-node detect and activate commands will query the SOAP server for detected and activated hardware information. Currently this is not the case.

In order to facilitate for slight differences in the API the user must specify which mode to compile for.

```
/* This is the only Sangoma transcoding header file necessary */
```

```
#ifndef SNGTC_NODE
#include <sng_tc/sngtc_node.h>
#else
#include <sng_tc/sng_tc.h>
#endif
```

The libsngtc-node library is provided in binary format on Linux, Windows for both 32bit and 64 bit architectures.

The main functions of libsngtc-node are:

- **Distribute all libsng-tc API calls**

The libsng-tc library calls are serialized and passed via SOAP TCP protocol to a SOAP Server that executes the commands via libsng-tc library.

- **Allow multiple applications to run over single hw resource**

Writing multiple applications over libsng-tc library is not possible since applications need to know if resources have been freed or card reset. Libsngtc-node library via SOAP server solves this issue.

- **Resource access over Internet**

All commands from libsng-tc to hw are limited to a single LAN, since all communication is Ethernet based. Thus, having an application control the hw from different lan is not possible. The libsngtc-node library uses SOAP calls to access the SOAP server, therefore resource can now be accessed from other Networks.

## SNGTC Media Transcoding Files

### Binary Libraries

- libsng-tc - low level library
- libsngtc-node - soap library for distributed operation

### libsng-tc: header files

- sng\_tc.h - main header file
- sng\_tc\_if.h - interface header file shared by the node
- sng\_tc\_utils.h - miscellaneous helper functions and defines
- sng\_tc\_version.h - version header file

### libsngtc-node: header files

- sngtc\_node.h - main node header file
- sngtc\_node\_utils.h - miscellaneous helper functions and defines

### Linux:

binary libraries are installed by default: /usr/lib/sngtc

header files are installed by default: /usr/include/sngtc

### Windows:

binary libraries are installed in test application directory

header files are installed in test application directory

## SNGTC Media Transcoding Functions - List

```
int sngtc_detect_init_modules(sngtc_init_cfg_t *cfg,  
                             int *detected);
```

```
int sngtc_activate_modules(sngtc_init_cfg_t *cfg,  
                           int *activated);
```

```
int sngtc_deactivate_modules(void);
```

```
int sngtc_create_transcoding_session  
    (struct sngtc_codec_request *codec_req,  
     struct sngtc_codec_reply *codec_reply,  
     int start_module);
```

```
int sngtc_free_transcoding_session  
    (struct sngtc_codec_reply *codec_reply);
```

## SNGTC Media Transcoding Functions

### sngtc\_detect\_init\_modules

```
int sngtc_detect_init_modules(sngtc_init_cfg_t *cfg, int *detected);
```

```
\brief Detect and group sngtc modules on specified lan interfaces.  
\param sngtc_init_cfg_t contains the list of eth interfaces that should  
        be probed for Sangoma transcoding cards.  
\param detected returns the number of detected modules  
\return 0 ok , non-zero error.
```

Detect and group found Sangoma transcoding modules.  
An ethernet broadcast is sent out over the LAN. If a Sangoma transcoding module is active it will respond. Each sngtc module detected is put in resource pool to be used by the application.

#### Arguments:

sngtc\_init\_cfg\_t \*cfg

Contains the list of ethernet interfaces that should be probed for Sangoma transcoding cards.

int \*detected

returns the number of detected modules.

#### Return code:

0 ok , non-zero error.

This function is performed only once on library startup.

## sngtc\_activate\_modules

```
int sngtc_activate_modules(sngtc_init_cfg_t *cfg, int *activated);
```

```
\brief Iterate over all detected modules and perform a full reset on  
      each module.  
\param sngtc_init_cfg_t same cfg structre used in detect_init fuction.  
\param activated returns the number of activated modules  
\return 0 ok , non-zero error.
```

Iterate over all detected modules and perform a full reset on each module. Check that firmware and configuration of each module is correct. Return number of successfully activated Sangoma transcoding modules.

Arguments:

sngtc\_init\_cfg\_t \*cfg

Initial configuration structure. Not really used by activate function, but left for future use.

int \*activated

Returns number of activated modules.

Return code:

0 ok, non-zero error

This function is performed only once on library startup.

## sngtc\_deactivate\_modules

```
int sngtc_deactivate_modules(void);
```

```
\brief Free all active and detected modules and release resources.  
\return 0 ok , non-zero error.
```

Free all active and detected modules and release resources.

This function is performed only once on library shutdown.

Return code:

0 ok, non-zero error

## sngtc\_create\_transcoding\_session

```
int sngtc_create_transcoding_session
    (struct sngtc_codec_request *codec_req,
     struct sngtc_codec_reply *codec_reply,
     int start_module);
```

\brief Create transcoding session. Each transcoding session creates a 2 RTP legs.

\param codec\_req contains codec information for each leg as well as host ip and port information.

\param codec\_reply structure is filled by the Sangoma transcoding module and contains the IP

\param start\_module can be used to select a specific Sangoma transcoding module.

\return 0 ok , non-zero failed.

Create transcoding session. Each transcoding session creates a 2 RTP legs. The codec\_req structure contains codec information for each leg as well as host ip and port information. Once the codec legs are created, the Sangoma transcode module will start sending RTP packets to the IP address provided by the codec\_req structure. The host is expected to create RTP sessions that will be listening to the incoming rtp traffic.

The codec\_reply structure is filled by the Sangoma transcoding module and contains the IP and port information that will be used by the host to transmit RTP data to Sangoma transcoding module.

start\_module can be used to select a specific Sangoma transcoding module. By default the value of start\_module is ZERO, indicating auto select.

For much more detailed information about Create RTP session function please refer to APPENDIX.

## sngtc\_free\_transcoding\_session

```
int sngtc_free_transcoding_session  
    (struct sngtc_codec_reply *codec_reply);
```

```
\brief Free transcoding session. The RTP data from Sangoma transcoding  
card will stop.  
\param codec_reply must be the same structure that was received from  
sngtc_create_transcoding_session  
\return 0 - ok, non-zero failed
```

Free transcoding session. The RTP data from Sangoma transcoding card will stop and resources on the Sangoma transcoding card will be freed related to that transcoding channel.

codec\_reply structure must be the same structure that was received from Sangoma transcoding module on session create. It contains the session id that is used by the library to indicate what session to free.

Return code: 0 ok , non-zero failed.

This function is thread safe and can be called from multiple threads at the same time.

## SNGTC Media Transcoding Structures

### sngtc\_init\_cfg

```
\brief Sangoma transcode init structure used to configure the transcode
    with initial parameters.
\typedef sngtc_init_cfg_t
```

```
typedef struct sngtc_init_cfg
{
    /* Library operation mode: LIBRARY, SOAP_CLIENT, SOAP_SERVER */
    int operation_mode;

    /* Host NIC configuration is only used in non soap_mode.
       List of structures each indicating the Ethernet interface
       information that should be used
       to probe for Sangoma transcoding cards. */
    uint32_t host_nic_vocallo_sz;
    sngtc_host_nic_vocallo_cfg_t
        host_nic_vocallo_cfg[SNGTC_MAX_HOST_VOCALLO_NIC];

    /* log function is used when library would like to print
       information. Each log comes with a log level for easy
       priority parsing */
    int (*log)( int level, char *fmt, ...);

    /* create rtp callback is used during create transcoding
       session call. As soon as there is enough information obtained
       by the library for each leg of the call, the create_rtp is
       called so that application can create its rtp session and
       start transmitting and receiving data */
    int (*create_rtp)
        (void * usr_priv,
            sngtc_codec_request_leg_t *codec_reg_leg,
            sngtc_codec_reply_leg_t* codec_reply_leg,
            void **rtp_fd);

    /* create_rtp_port is called during the create transcoding
       session. In some applications the rtp port that will be used
       to create rtp session must be obtained earlier. The host_ip is
       the ip address of the host application, which can be used in
       rtp hunting procedures. */
    int (*create_rtp_port)(void * usr_priv,
            uint32_t host_ip,
            uint32_t *rtp_port,
            void **rtp_fd);

    /* destroy_rtp callback is called during free transcoding
       session call. At this time host application is able to free
       its rtp session resources */
    int (*destroy_rtp)(void * usr_priv, void *fd);
} sngtc_init_cfg_t;
```

## sngtc\_codec\_request

\brief Used to configure the whole call session that includes 2 legs of the call.

\typedef sngtc\_codec\_request\_t

```
typedef struct sngtc_codec_request
{
    /*!< user void pointer used in user callbacks */
    void          *usr_priv;

    /*!< identifies a session with a tag. Used to shutdown all
        sessions via tag */
    uint64_t     tag;

    /*!< enable or disable rtcp in sngtc hardware */
    uint8_t     rtcp_enable;

    /*!< leg a configuration */
    struct sngtc_codec_request_leg a;

    /*!< leg b configuration */
    struct sngtc_codec_request_leg b;
} sngtc_codec_request_t;
```

## sngtc\_codec\_request\_leg

\brief Used to configure codec information for each leg of the call.

\typedef sngtc\_codec\_request\_leg\_t

```
typedef struct sngtc_codec_request_leg
{
    /*!< codec id: enum sngtc_codec_definition */
    uint32_t     codec_id;

    /*!< codec ms parameter in milliseconds */
    uint32_t     ms;

    /*!< ip address of a host */
    uint32_t     host_ip;

    /*!< ip address of a host netmask */
    uint32_t     host_netmask;

    /*!< host rtp port */
    uint32_t     host_udp_port;
} sngtc_codec_request_leg_t;
```

## sngtc\_codec\_reply

\brief Codec cfg from the sngtc module for the whole session: 2 legs,  
as the response codec request.

\typedef sngtc\_codec\_reply\_t

**typedef struct sngtc\_codec\_reply**

```
{  
    /*!< id to identify sngtc module */  
    uint32_t    codec_module_session_idx;  
  
    /*!< id to identify rtp session inside the sngtc module */  
    uint32_t    codec_rtp_session_idx;  
  
    /*!< per leg sngtc codec information */  
    struct sngtc_codec_reply_leg a;  
  
    /*!< per leg sngtc codec information */  
    struct sngtc_codec_reply_leg b;  
  
    /*!< For library use: used during user callbacks */  
    void        *tx_fd;  
  
    /*!< For library use: used during user callbacks */  
    uint32_t    tx_iana;  
  
    /*!< For library use: used during user callbacks */  
    void        *rx_fd;  
  
    /*!< For library use: used during user callbacks */  
    uint32_t    rx_iana;  
  
    /*!< used in soap mode to provide return code */  
    int32_t        result;  
}  
 } sngtc_codec_reply_t;
```

## sngtc\_codec\_reply\_leg

```
\brief Per leg codec configuration obtained from the Sangoma  
transcoding module as a response to the codec request.  
\typedef sngtc_codec_reply_leg_t
```

```
typedef struct sngtc_codec_reply_leg  
{  
    /*!< ip address of sngtc rtp session */  
    uint32_t    codec_ip;  
  
    /*!< netmask of sngtc rtp session */  
    uint32_t    codec_netmask;  
  
    /*!< rtp port of the sngtc rtp session */  
    uint32_t    codec_udp_port;  
  
    /*!< host ip info obtained from codec_request structure */  
    uint32_t    host_ip;  
  
    /*!< host netmask info obtained from codec_request structure */  
    uint32_t    host_netmask;  
  
    /*!< host rtp port obtained from codec_request structure */  
    uint32_t    host_udp_port;  
  
    /*!< iana code used in the rtp header: payload type */  
    uint32_t    iana_code;  
  
}sngtc_codec_reply_leg_t;
```

## SNGTC Sample Code Library

### Library Initialization

```
/* This is the only Sangoma transcoding header file necessary */

/* When working with libsnt-tc use #include <sng_tc/sng_tc.h>
   When working with libsngtc-node use #include <sng_tc/sng_tc.h>

   The SNGTC_NODE define must be define on application compile time.
*/

#ifdef SNGTC_NODE
    #include <sng_tc/sngtc_node.h>
#else
    #include <sng_tc/sng_tc.h>
#endif

/* Initialize the init_cfg structure.
   This is done one at the start of the program. Please refer to the
   structure documentation above.

   In this example, we will configure the init_cfg structure for a
   SINGLE ethernet interface. In case of multiple Sangoma
   transcoding cards, there could be multiple Ethernet interfaces that
   user would configure in init_cfg so that library could auto detect
   Sangoma transcoding hw on all configured Ethernet interfaces.

   Note that in NODE mode there are not Ethernet interfaces to detect
   since SOAP server already did this.

*/

sngtc_init_cfg_t init_cfg;

memset(&init_cfg,0,sizeof(init_cfg));

init_cfg.log = log_printf;
init_cfg.create_rtp = create_rtp;
init_cfg.create_rtp_port = create_rtp_port;
init_cfg.destroy_rtp = destroy_rtp;
init_cfg.operation_mode = SNGTC_MODE_LIBRARY;
```

```
#ifndef SNGTC_NODE

/* number of Ethernet devices to query. Hardcoded to 1 */

/* Ethernet detection is only done in libsng-tc mode.
   It is not done in NODE mode since SOAP server went through
   this processes already */

max_ethernet_interfaces=1

for (i=0;i<max_ethernet_interfaces;i++) {

    init_cfg.host_nic_vocallo_sz = i+1;

    /* 00-1B-FC-DA-E6-A6 is the MAC of local Ethernet device on the
       host */
    sprintf((char*)init_cfg.host_nic_vocallo_cfg[i].host_mac.mac_str,
            "00-1B-FC-DA-E6-A6");

    /* Specify local Ethernet IP address and NetMask */
    sngtc_codec_ipv4_str_hex("192.168.1.32",
        &init_cfg.host_nic_vocallo_cfg[i].host_ip);

    sngtc_codec_ipv4_str_hex("255.255.255.0",
        &init_cfg.host_nic_vocallo_cfg[i].host_ip_netmask);

    /* Specify Sangoma transcoding HW IP address and netmask */
    sngtc_codec_ipv4_str_hex("192.168.1.183",
        &init_cfg.host_nic_vocallo_cfg[i].vocallo_ip);

    sngtc_codec_ipv4_str_hex("255.255.255.0",
        &init_cfg.host_nic_vocallo_cfg[i].vocallo_ip_netmask);

    /* Specify the gateway IP address related to Sangoma
       Transcoding HW */

    sngtc_codec_ipv4_str_hex("192.168.1.1",
        &init_cfg.host_nic_vocallo_cfg[i].gateway_ip);

    /* Specify the base rtp port that Sangoma transcoding hw will use
       on rtp session create. This value will be used as base offset. */
    init_cfg.host_nic_vocallo_cfg[i].vocallo_base_udp_port = rtp_port;

    /* Enable (1) or Disable (0) silence suppression */
    init_cfg.host_nic_vocallo_cfg[i].silence_suppression=0;

}

#endif

/* Initialize the library and auto detect Sangoma
   transcoding hardware that is connected to Ethernet interfaces
   specified above. The detected variable will return number of
   detected Sangoma transcoding modules. */
```

```
err=sngtc_detect_init_modules(&init_cfg,&detected));
if (err) {
    /* Failed to detect any modules */
    exit(1);
}

if (detected != expected_detected) {
    /* If one knows that there should be 2 detected devices and the
       detected variable only returns 1. Then user can handle this
       condition */

    exit(1);
}

/* Activate all detected modules. This call will check each module
   configuration and operational info. Activated variable returns
   number of activated devices */

err= sngtc_activate_modules(&init_cfg,&activated);
if (err) {
    /* Failed to activate any modules */
    exit(1);
}

if (detected != activated) {
    /* In ideal scenario number of detected devices should be equal
       to number of activated devices. At this point user can take
       action */
    exit(1);
}

/* At this point the system is ready and all modules are awaiting
   resource requests. In real world application, there could be a
   thread associated with each codec request. In this example we will
   create a single codec request.

   Both libsng-tc and libsngtc-node are thread safe libraries so no
   extra locking is needed to synchronize codec request and codec free
   calls. */
```

## Library Operation

```
/* In this example private_codec_t is a user structure that contains
   codec_request and codec_reply structures as well as all the rtp
   session information of the user application that is needed per
   codec session */

private_codec_t priv_codec;

/* Initialize the codec_request structure. In this example we will
   create G711 to G729 transcoding session */

priv_codec.codec_request.usr_priv=priv_codec;

/* Initialize LEG A: with codec information */
priv_codec.codec_request.a.codec_id=SNGTC_CODEC_PCMU;
priv_codec.codec_request.a.ms=20; /* 20 ms */

/* Initialize LEG B: with codec information */
priv_codec->codec_request.b.codec_id=SNGTC_CODEC_G729AB;
priv_codec->codec_request.b.ms=ms;

err=sngtc_create_transcoding_session
    (&priv_codec->codec_request, &priv_codec->codec_reply,0);

if (err) {
    /* Error: Failed to create transcoding session */
    exit (1);
}

/* At this point the hw has been configured for a 2 transcoding legs.
   G711 and G729 at 20ms with silence suppression off. During the
   sngtc_create_transcoding_session call, the libsng-tc library called
   the create_rtp and create_rtp_port callback functions in order for
   the user application to create its RTP session based on IP
   information obtained from the Sangoma transcoding hardware */

/* When the application call is finished, and its time to stop the
   codec session..... */

err=sngtc_free_transcoding_session(&priv_codec->codec_reply);
if (err) {
    /* Failed to free transcoding session. Possible reason is
       user error ☺ or in SOAP mode the server has gone down.

       There is no good way to recover from this error
    */
    exit(1);
}
```

## Library Callbacks

```

/* Application specific way to obtain the free host RTP port.
   Pass the obtained PORT back to the library via p_rtp_port.
   In some circumstances rtp session must be created before
   being able to hunt for the port. In that case rtp_fd is used
   to store the rtp session file descriptor. */
static int create_rtp_port
    (void *usr_priv, uint32_t host_ip,
     uint32_t *p_rtp_port, void **rtp_fd)
{
    *p_rtp_port = globals.base_port;
    *rtp_fd = (void *) (long) globals.base_port;
    globals.base_port += 2;
    return 0;
}

/* Application specific way to create RTP session.
   Pass the RTP session file descriptor to the library via rtp_fd.
   The codec_reply leg contains all ip information from both
   host and hw. Please review the codec_reply structure above. */
static int create_rtp_socket
    (void *usr_priv, sngtc_codec_request_leg_t *codec_req,
     sngtc_codec_reply_leg_t *codec_reply, void **rtp_fd)
{
    RtpSession *session = NULL;
    uint32_t local_port = codec_reply->host_udp_port;

    ip_to_str(codec_reply->host_ip, bindip_str);

    session = rtp_session_new(RTP_SESSION_SENDRECV);

    rtp_session_set_local_addr(session,
                               bindip_str, local_port);

    rtp_session_set_connected_mode(session, TRUE);
    *rtp_fd = session;
    return 0;
}

/* Application specific way to destroy the RTP session.
   Library gives back the RTP session file descriptor fd */
static int destroy_rtp_socket(void *usr_priv, void *fd) {
    RtpSession *session = fd;
    rtp_session_destroy(session);
    return 0;
}

```

## Library Shutdown

```
/* On shutdown there is one call to free up all library resources and  
   reset the card */
```

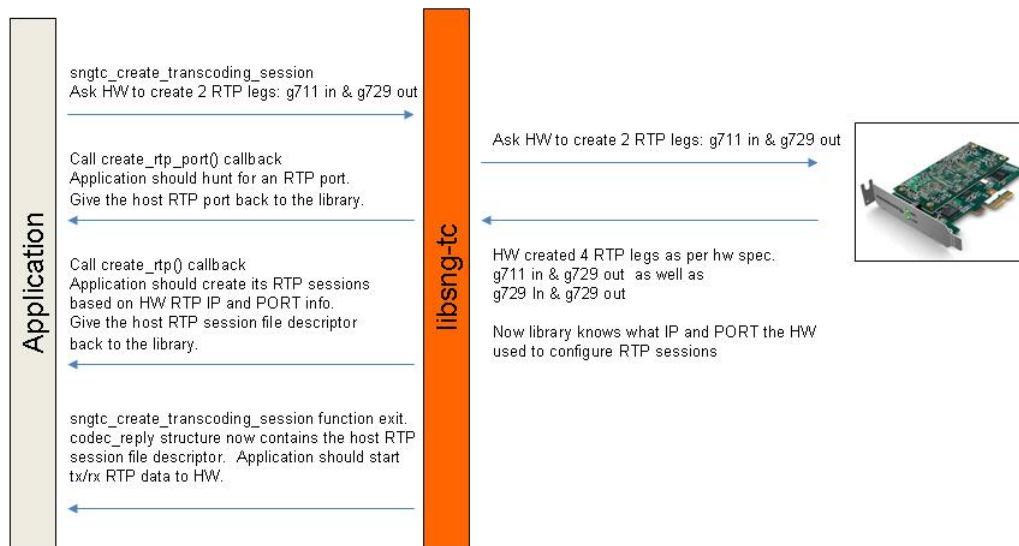
```
err=sngtc_deactivate_modules();
```

## APPENDIX

## SNGTC Create RTP Session – New

11/2/2010

## SNGTC: Create RTP Session - new



The Sangoma transcoding hw is designed in such way that each transcoding resource is symmetrical. Thus each audio transcoding resource has 4 RTP legs associated with it.

In case of G711 to G729 transcoding there will be:

- 2 RTP session: G711 In and G729 Out
- 2 RTP session: G729 In and G711 Out

Even though a codec might request only G711 to G729 path, the hardware always creates 2 transcoding paths, or 4 RTP sessions, because the transcoding resources are symmetrical in nature.

The `sngtc_create_transcoding_session()` function hides this fact from the application, since in most scenarios there will always be 2 legs on every call, each having 2 RTP sessions, and each leg would need to be transcoded.

The `codec_req` structure contains codec information for single path or 2 RTP sessions (eg: G711 In an G729 out), as well as host ip and port information. In most codec applications each codec path is setup separately.

As specified above, as soon as G711 to G729 path is created, the hw automatically creates the reverse path of G729 to G711. The library knows this and marks the RTP session so on next call to `sngtc_create_transcoding_session` if the request is for G729 to G711 the same hw rtp session is reused but tx and rx file descriptors flipped.

Once the codec legs are created, the Sangoma transcode module will start sending RTP packets to the IP address provided by the `codec_req` structure. The application is expected to create RTP sessions that will be listening to the incoming rtp traffic.

The `codec_reply` structure is filled by the Sangoma transcoding module and contains the IP and port information that will be used by the application to transmit RTP data to Sangoma transcoding module.

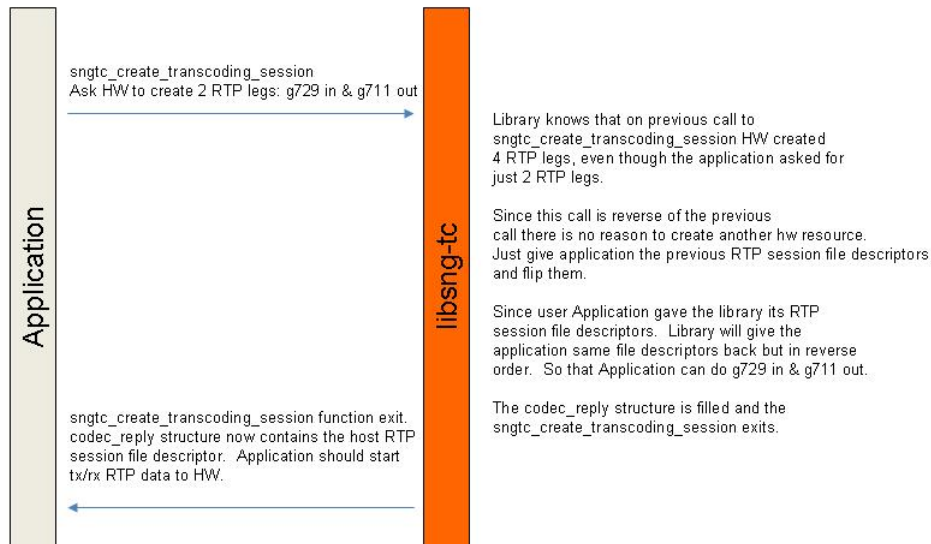
Once the `codec_reply` structure is filled by the Sangoma transcoding module, the library calls the user application via "create\_rtp\_port" and "create\_rtp" callback in order to for the application to create its own RTP session and start communicating to the Sangoma transcoding hardware.

After successful `create_rtp` callback, the RTP session file descriptor created in `create_rtp` callback is saved in `codec_reply` structure, and the `sngtc_create_transcoding_session` function exists with a return code.

After the `sngtc_create_transcoding_session` function the application can start to poll on the RTP session file descriptor and tx/rx RTP packets to and from the Sangoma transcoding hardware.

## SNGTC Create RTP Session – Cached

11/2/2010

**SNGTC: Create RTP Session - cached**

Note that from applications point of view `sngtc_create_transcoding_session` created only one codec path.

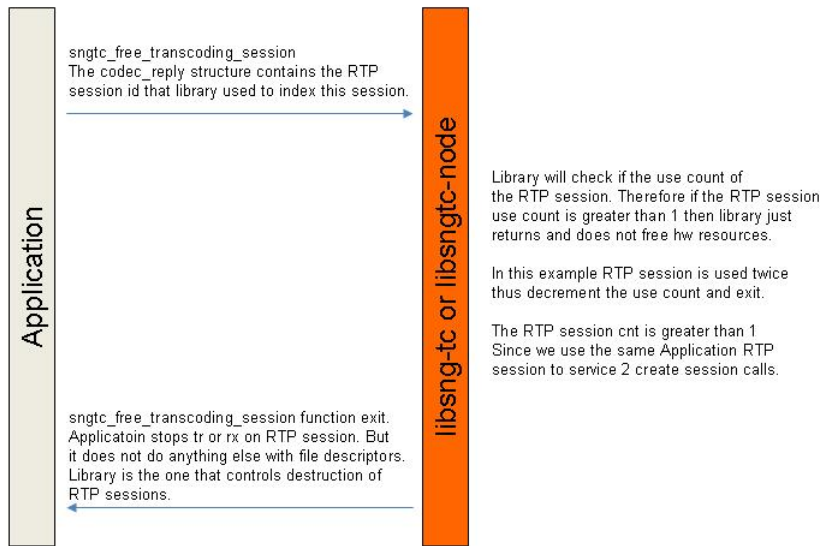
However hw has created 2 codec paths. Therefore on next call to `sngtc_create_transcoding_session` if the `codec_req` path matches the reverse path of the previous request, the library will not ask hardware to create another resource, instead it will reverse the tx and rx file descriptors and will give the application the same RTP session that it has created in previous session `create_rtp()` callback.

Since it's the library that controls when application RTP sessions are created and destroyed, the application needs to trust the library and use the file descriptors only to tx and rx. Application must not try to close the RTP session file descriptors itself, otherwise another leg of the call might be corrupted.

## SNGTC Free RTP Session – Cached

11/2/2010

### SNGTC: Free RTP Session - cached



## SNGTC Free RTP Session – New

11/2/2010

### SNGTC: Free RTP Session - new

