



## Cisco TSP Media Driver

---

Cisco Media Driver introduces a new and innovative way for TAPI-based applications to provide media interaction such as play announcements, record calls, and so on.

Cisco TSP 8.0(1) includes support for both Cisco Media Driver and Cisco Wave Driver, but only one driver can be active at any given time.

Cisco Media Driver offers several advantages:

- **Simplified Installation and Management**—Cisco Media Driver configuration can be completed through the Cisco TSP Installation Wizard. Channel and port settings are consistently and automatically applied to all configured TSP instances.
  - **Performance and Scalability**—Cisco Media Driver can scale to support up to 1000 configured ports with hundreds of simultaneously active media channels. Refer to the application vendor's Installation Guide to determine the number of channels supported by the TAPI application.
  - **Codec Support**—Cisco Media Driver supports 8KHz, 16-bit PCM, G.711 a-law, G.711 u-law natively. Additionally, G.729a can be supported when pass-through mode is enabled.
  - **Reliability**—Cisco Media Driver runs as an independent process, similar to Windows applications, providing greater application stability and reliability. Creating and debugging media applications is now much easier.
- [Cisco Rtp Library Components, on page 1](#)
  - [TAPI Application Support, on page 3](#)
  - [EpAPI Functions, on page 6](#)
  - [EpApi Error Codes, on page 21](#)
  - [Callback Function, on page 22](#)
  - [Data Structures, on page 23](#)
  - [Trace Options, on page 25](#)
  - [Known Problems or Limitations, on page 26](#)

## Cisco Rtp Library Components

### Header Files

The following header files contain declaration of all functions, data structures, etc. exposed by Cisco Rtp Library.

- ciscortpapi.h
- ciscortpbase.h
- ciscortpcbes.h
- ciscortpcodec.h
- ciscortperr.h
- ciscortpep.h
- ciscortpip.h

In order to use Cisco Rtp Library functionality a typical application would only need to explicitly include ciscortpapi.h and ciscortpep.h files.

### Import Library

The following import library has to be linked with an application in order to use Cisco Rtp Library functionality:

- cmrtplib.lib

### DLLs

The following DLLs are installed as a part of CiscoTSP plug-in and used by Cisco Rtp Library:

Windows 32bit OS (x86):

- ciscortplib.dll
- ciscortpmon.dll
- ciscortpg711a.dll
- ciscortpg711u.dll
- ciscortpg729.dll
- ciscortppcm16.dll

Windows 64bit OS (x64):

- ciscortplib64.dll
- ciscortpmon64.dll
- ciscortpg711a64.dll
- ciscortpg711u64.dll
- ciscortpg72964.dll
- ciscortppcm1664.dll

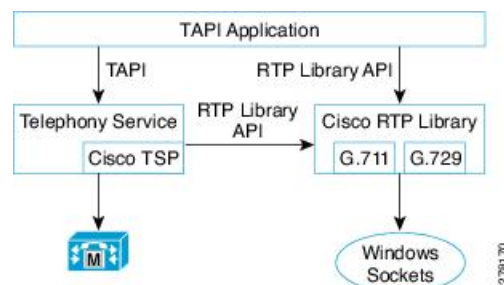
# TAPI Application Support

## CiscoTSP and Cisco Rtp Library Interaction

In order to allow TAPI applications to associate TAPI line device with Rtp Library media endpoints Cisco TSP implements two new device classes: ciscowave/in and ciscowave/out. If TAPI line device is capable to terminate media by means of Cisco Rtp Library, an application can use ciscowave/in and ciscowave/out device class names in the TAPI lineGetID() function to obtain associated media device identifiers. Media device identifier can be used in Cisco Rtp Library APIs to create media endpoints and manipulate media on a corresponding TAPI line device.

The following figure shows high level view of TAPI application which uses Cisco TAPI service provider and Cisco Rtp Library functionalities.

**Figure 1: TAPI Application with Cisco Components**



## Codec Advertisement

Cisco Media Driver devices advertise G.711 support natively. Cisco Unified CM automatically invokes Media Termination Points (MTPs) when needed to provide transcoding (see Example 1). If MTPs are not configured and transcoding is required, call setup fails (see Example 2).

### Example 1

1. G729PassThrough set to OFF (default).
2. TSP application registers CTI port 1.
3. CTI port 1 advertises G.711 support (default).
4. Unified CM is configured with MTPs, which can be used if transcoding is needed.
5. CTI port 1 calls Device 1000.
6. Device 1000 only supports G.729, so an MTP is inserted to provide transcoding.

### Example 2

1. G729PassThrough set to OFF (default).

2. TSP application registers CTI port 1.
3. CTI port 1 advertises G.711 support (default).
4. Unified CM is not configured with MTPs for transcoding.
5. CTI port 1 calls Device 1000.
6. Device 1000 only supports G.729 and no MTPs are available, so call setup fails.

Applications which natively support G.729 can change the default codec advertisement by setting the G729PassThrough registry option to ON (1).

The TSP application is then responsible for playing the appropriate media file (G.711 or G.729) based on the compatible codecs supported by the Device receiving the media (see Example 3 below).

The Registry key can be found at:

- Windows XP: HKEY\_Local\_Machine/Software/Cisco Systems, Inc./ RtpLib/G729PassThrough
- Windows Vista: HKEY\_USERS\S-1-5-20\Software\Cisco Systems, Inc.\ RtpLib\G729PassThrough

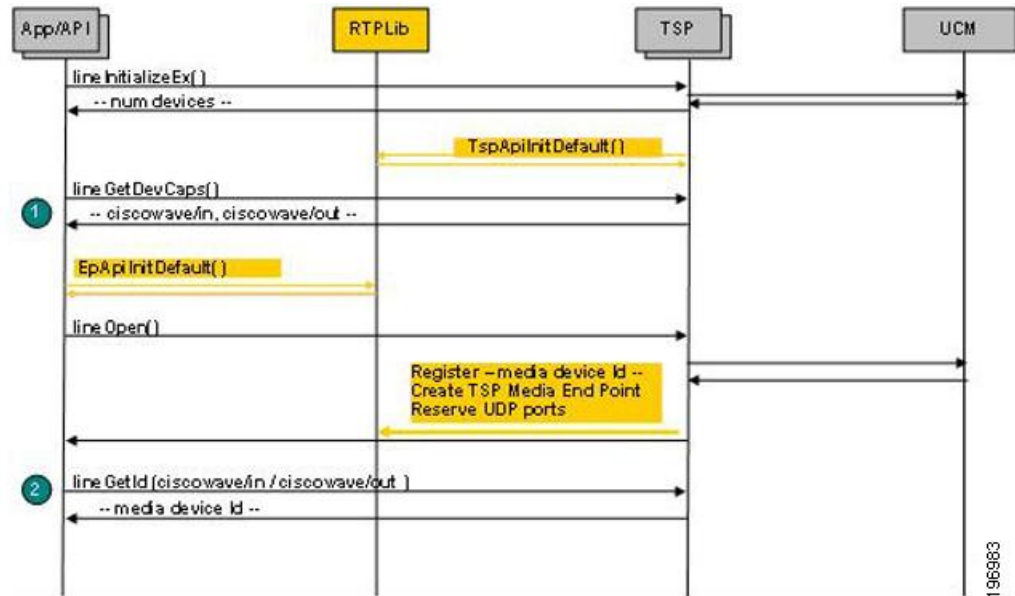
### Example 3

1. G729PassThrough set to ON.
2. TSP application registers CTI port 1.
3. CTI port 1 advertises G.711 and G.729 support.
4. Unified CM is not configured with MTPs for transcoding.
5. CTI port 1 calls Device 1000.
6. Device 1000 only supports G.729, so the application plays the appropriate G.729 media file.

## Typical TAPI Application Message Flow

The message flow in the following figure is described in steps 1 and 2.

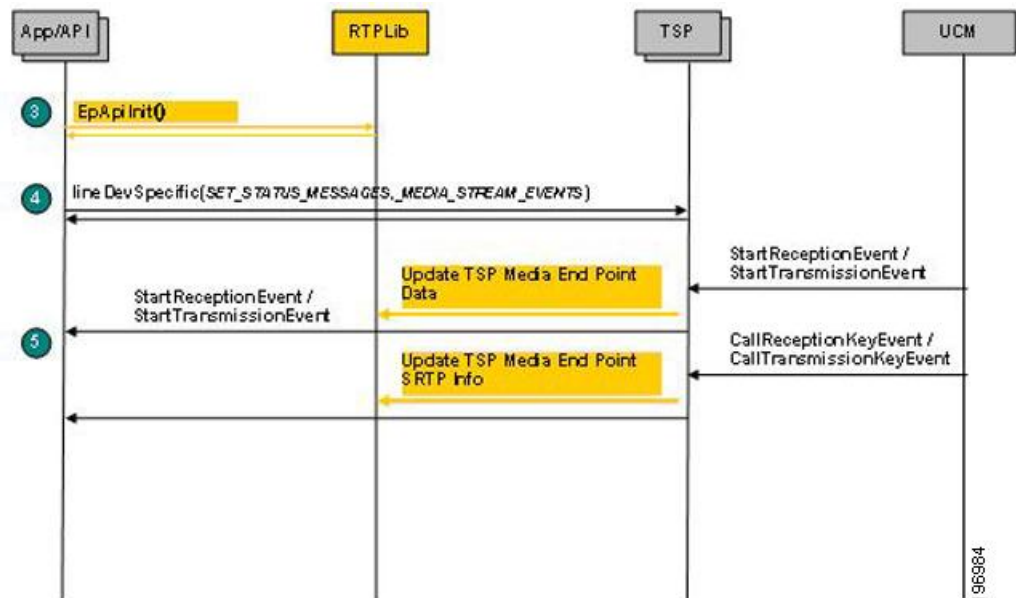
Figure 2: Typical TAPI Application Message Flow 1



1. Initialize TAPI, get LINEINFO for available line devices, find devices which are capable of using Cisco Rtp Library functionalities
2. Get media device identifier associated with a particular line device

The message flow in the following figure is described in steps 3 to 5.

Figure 3: Typical TAPI Application Message Flow 2

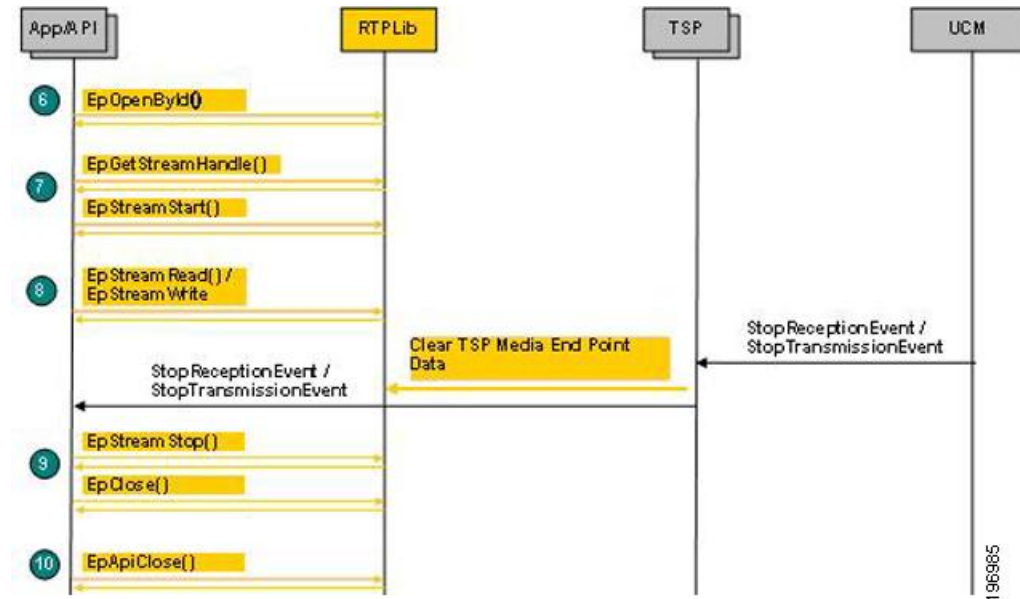


3. Initialize Rtp Library
4. Subscribe for media stream events for relevant devices using Cisco lineDevSpecific extension

## 5. Start monitoring media events

The message flow in following figure is described in steps 6 to 10.

**Figure 4: Typical TAPI Application Message Flow 3**



6. Create media endpoint
7. Get in/out stream handle and start data streaming
8. Receive / transmit data
9. Stop data streaming, close endpoint
10. Close EpAPI before exiting the program

# EpAPI Functions

## EpApiInit

Initializes EpApi and Rtp Library.

### Syntax

```

CMAPI bool    EpApiInit (
PRTPLIBTRACE  pTraceCallback,
USHORT        portRangeStart,
USHORT        numPorts,
int           IPAddressFamily,
PRTPADDR      pDefaultRtpAddr
);
  
```

**Parameters****pTraceCallback**

Pointer to an application callback function to be called with the trace record data passed to it.

**portRangeStart**

First port number in the continuous range of UDP ports (port pool) which can be used to create endpoints.

**numPorts**

Number of ports in the UDP port range (port pool) which can be used to create endpoints.

**IPAddressFamily**

IP address family to be used by Rtp Library to create endpoints can be set to:

- AF\_UNSPEC: both AF\_INET and AF\_INET6 can be used.
- AF\_INET: AF\_INET only can be used.
- AF\_INET6: AF\_INET6 only can be used.

This settings can be overwritten by the pDefaultRtpAddr parameter.

**pDefaultRtpAddr**

IP address to be used use by Rtp Library to create endpoints. If not NULL, only this address will be used.

**Return Value**

If no errors occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_ADDR_NOTAVAIL	Unable to create endpoint with specified IP address family
EP_ERR_PARAM_INVALID	The following describes a possible cause of the error: Invalid number of UDP ports
RTP_ERR_INITALREADY	Already initialized
RTP_ERR_TIMER_NOTAVAIL	Unable to create high resolution timer

**Remarks**

An error code can be set even when EpApiInit returns true. In some cases a default action / value can be assumed even if a parameter or registry settings is invalid. In those cases EpApiInit returns true but also set a proper error code to indicate an issue.

## EpApiInitByDefault

Initializes EpApi and Rtp Library with default settings.

### Syntax

```
CMAPI bool EpApiInitByDefault (
    PRTPLIBTRACEpTraceCallback,
);
```

### Parameters

#### pTraceCallback

Pointer to an application callback function to be called with the trace record data passed to it.

### Return Value

If no errors occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
RTP_ERR_INITALREADY	Already initialized

### Remarks

Rtp Library will be initialized as if registry is set as follows:

- UDPPortRangeStart = 50000
- UDPPortRangeEnd = 50999

## EpApiClose

Closes EpApi.

### Syntax

```
CMAPI bool EpApiClose ();
```

### Return Value

If no errors occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.

### Remarks

As a result of this function execution all active sessions, connections and streams will be terminated, timers closed and all data freed.



## EpLocalAddressGetAll

Returns an array of RTPADDR structures which contain local IP addresses available for use by Rtp Library.

### Syntax

```
CMAPI int EpLocalAddressPortGetAll(  
    RTPADDR pBuffer,  
    int *    pBufSize  
);
```

### Parameters

#### pBuffer

Pointer to a memory buffer to fill in with array of RTPADDR structures or NULL.

#### pBufSize

- IN—Length of the buffer (in bytes), pointed to by pBuffer.
- OUT—Space in the buffer used or required.

### Return Value

If no errors occurs, this function returns a number of available local IP addresses and an array of RTPADDR structures in the pBuffer.

If pBuffer parameter value is NULL, the function returns the number of available local IP addresses and the pBufSize will contain the buffer size required for the RTPADDR structure array.

If an error occurs, 0 is returned and a specific error code can be retrieved by calling EpApiGetLastError. In case of EP\_ERR\_PARAM\_INVALID error, pBufSize will contain the size of the required buffer.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_PARAM_INVALID	Buffer is not large enough.

## EpLocalAddressPortGet

Reserves port from the port pool and returns it together with a local IP address.

### Syntax

```
CMAPI RTPADDR EpLocalAddressPortGet();
```

### Return Value

If no errors occurs, this function returns pointer to RTPADDR structure with the first (or default) local IP address used by Rtp Library and reserved UDP port number.

If an error occurs, NULL is returned and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
RTP_ERR_PORT_NOTAVAIL	No UDP port available.

## EpLocalAddressPortGetByFamily

Reserves port from the port pool and returns it alone with a local IP address for the specified family.

### Syntax

```
CMAPI PRTPADDR EpLocalAddressPortGetByFamily(
    int          IPAddressFamily
);
```

### Parameters

#### IPAddressFamily

IP address family: AF\_INET or AF\_INET6

Returns: Pointer to RTPADDR structure or NULL.

### Return Value

If no errors occurs, this function returns pointer to RTPADDR structure with the first local IP address for the specified family used by Rtp Library and reserved UDP port number. If an error occurs, NULL is returned and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
RTP_ERR_ADDR_NOTAVAIL	No IP address available for specified family
RTP_ERR_PORT_NOTAVAIL	No UDP port available.

## EpLocalAddressPortGetByIdx

Reserves UDP port from the Rtp Library port pool and returns it in the RTPADDR data structure alone with the IP address of the network interface card specified by the index parameter.

### Syntax

```
CMAPI PRTPADDR EpLocalAddrPortGetByIdx (
    int          index
);
```

**Parameters****index**

Index in the list of available local network addresses returned by EpLocalAddressGetAll function call.

**Return Value**

If no error occurs, this function returns pointer to RTPADDR structure which contains local IP address and reserved UDP port number. If an error occurs, NULL is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_PARAM_INVALID	Invalid index value.
RTP_ERR_PORT_NOTAVAIL	No UDP port available.

**Remarks**

List of available local network addresses can be obtained by EpLocalAddressGetAll function call.

## EpLocalAddrPortFree

Returns local UDP port previously reserved by EpLocalAddressGet, EpLocalAddressGetByIdx or EpLocalAddressGetByFamily back to the port pool.

**Syntax**

```
CMAPI bool EpLocalAddrPortFree (
    RTPADDR pLocalAddrPort
);
```

**Parameters****pLocalAddrPort**

Pointer to RTPADDR data structure which contains port number of previously reserved local UDP port.

**Return Value**

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.

**Remarks**

Local UDP could is reserved by EpLocalAddressGet, EpLocalAddressGetByIdx or EpLocalAddressGetByFamily.

## EpOpenById

Creates media endpoint based on TSP data associated with the specified media device identifier.

**Syntax**

```
CMAPI_HANDLE EpOpenById (
    DWORD          deviceId,
    StreamDirection streamDir,
    PRTPDATACALLBACK pCallback
);
```

**Parameters****deviceId**

Media device identifier obtained by calling TAPI lineGetID() for ciscowave/in or ciscowave/out device class.

**streamDir**

Stream direction. This parameter can be one of the following values:

- ToApp
- ToNwk
- Both

**pCallback**

Pointer to an application callback function to be called when data buffer is received/sent or an error occurred.

**Return Value**

If no errors occurs, this function returns a handle which can be used to reference the endpoint. If an error occurs, NULL is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_PARAM_INVALID	The following describes a possible cause of the error: <ul style="list-style-type: none"><li>• Specified device identifier is invalid.</li><li>• Required data associated with device identifier data is missing</li><li>• Specified stream direction is invalid</li></ul>

**Remarks**

Endpoint is created based on a data associated by TSP with the deviceId.

## EpClose

Close endpoint created by EpOpen.

**Syntax**

```
CMAPI bool EpClose (
    HANDLE    hEp
);
```

**Parameters****hEp**

Endpoint handle returned by EpOpen.

**Return Value**

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified endpoint handle is invalid.

## EpGetStreamHandle

Returns endpoint stream handle for a specified stream type and direction

**Syntax**

```
CMAPI HANDLE EpGetStreamHandle (
    HANDLE    hEp,
    StreamType    streamType,
    StreamDirection    streamDir
);
```

**Parameters****hEp**

Endpoint handle returned by EpOpen.

**streamType**

Stream type. This parameter can be one of the following values:

- STREAM\_TYPE\_AUDIO
- STREAM\_TYPE\_VIDEO

**streamDir**

Stream direction. This parameter can be one of the following values:

- ToApp
- ToNwk

**Return Value**

If no errors occurs, this function returns stream handle which can be used to reference the stream. If an error occurs, NULL is returned, and a specific error code can be retrieved by calling EpApiGetLastError

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified endpoint handle is invalid.
EP_ERR_PARAM_INVALID	Specified stream type or direction is invalid.

## EpStreamStart

Enables data flow on a specified stream

**Syntax**

```
CMAPI bool EpStreamStart (
    HANDLE          hStream,
    PRTPDATACALLBACK pCallback
);
```

**Parameters****hStream**

Stream handle returned by EpGetStreamHandle.

**pCallback**

Pointer to an application callback function to be called when data buffer is received/sent or an error occurred.

**Return Value**

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified endpoint handle is invalid.
EP_ERR_ADDR_INUSE	Address (protocol-IPaddress-port) is already in use.

**Remarks**

EpStreamStart() should be explicitly called by an application in order to stream data flow (open socket, port). It is not done implicitly by the Rtp Library as it was done before by the Cisco kernel mode wave driver.

## EpStreamStop

Disables data flow on a specified stream.

**Syntax**

```
CMAPI bool EpStreamStop (
    HANDLE    hStream
);
```

**Parameters****hStream**

Stream handle returned by EpGetStreamHandle.

**Return Value**

If no errors occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified stream handle is invalid.

**Remarks**

EpStreamStop() should be explicitly called by an application in order to disable stream data flow. It is not done implicitly by the Rtp Library as it was done before by the Cisco kernel mode wave driver.

## EpStreamRead

Read data from a stream.

**Syntax**

```
CMAPI bool EpStreamRead (
    HANDLE          hStream,
    PCHAR           pBuffer,
    int             bufLen,
    PVOID           pAppData,
    PRTPDATAcallback pCallback
);
```

**Parameters****hStream**

Stream handle returned by EpGetStreamHandle.

**pBuffer**

Pointer to a buffer for incoming data.

**bufLen**

Buffer size.

**pAppData**

Pointer to an application data area. It will be associated with the buffer and will be passed back to the application callback function as the pAppData parameter.

**pCallback**

Pointer to an application callback function to be called when data buffer is received or an error occurred.

**Return Value**

If no errors occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified stream handle is invalid.

## EpStreamWrite

Write data to a stream.

**Syntax**

```
CMAPI bool EpStreamWrite (
    HANDLE          hStream,
    PCHAR           pBuffer,
    int             bufLen,
    PVOID           pAppData,
```



```
PRTPDATACALLBACK pCallback
);
```

### Parameters

#### hStream

Stream handle returned by EpGetStreamHandle.

#### pBuffer

Pointer to a buffer which contains data.

#### bufLen

Data length

#### pAppData

Pointer to an application data area. It will be associated with the buffer and will be passed back to the application callback function as the pAppData parameter.

#### pCallback

Pointer to an application callback function to be called when data buffer has been written or an error occurred.

### Return Value

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified stream handle is invalid.

## EpStreamCodecInGet

Returns stream inbound codec format information.

### Syntax

```
CMAPI bool EpStreamCodecInGet (
    HANDLE          hStream,
    PWAVEFORMATEX  pWaveFormat
);
```

### Parameters

#### hStream

Stream handle returned by EpGetStreamHandle.

**pWaveFormat**

Pointer to a WAVEFORMATEX data structure. Upon successful completion of the request this structure is filled with stream inbound codec format data.

**Return Value**

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified stream handle is invalid.

## EpStreamCodecInSet

Sets stream inbound codec format.

**Syntax**

```
CMAPI bool EpStreamCodecInSet (
    HANDLE          hStream,
    PWAVEFORMATEX   pWaveFormat,
    ULONG           pktSizeMs
);
```

**Parameters****hStream**

Stream handle returned by EpGetStreamHandle.

**pWaveFormat**

Pointer to a WAVEFORMATEX data structure which contains codec information.

**pktSizeMs**

Packet size in milliseconds. If value 0 (zero) is specified a default value (20) is used.

**Return Value**

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified stream handle is invalid.

## EpStreamCodecOutGet

Returns stream outbound codec format information.

### Syntax

```
CMAPI bool EpStreamCodecOutGet (  
    HANDLE      hStream,  
    PWAVEFORMATEX pWaveFormat  
);
```

### Parameters

#### hStream

Stream handle returned by EpGetStreamHandle.

#### pWaveFormat

Pointer to a WAVEFORMATEX data structure. Upon successful completion of the request this structure is filled with stream outbound codec format data.

### Return Value

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified stream handle is invalid.

## EpStreamCodecOutSet

Sets stream outbound codec format.

### Syntax

```
CMAPI bool EpStreamCodecInSet (  
    HANDLE      hStream,  
    PWAVEFORMATEX pWaveFormat,  
    ULONG      pktSizeMs  
);
```

### Parameters

#### hStream

Stream handle returned by EpGetStreamHandle.

#### pWaveFormat

Pointer to a WAVEFORMATEX data structure which contains codec information.

**pktSizeMs**

Packet size in milliseconds. If value 0 (zero) is specified a default value (20) is used.

**Return Value**

If no error occurs, this function returns true. If an error occurs, false is returned, and a specific error code can be retrieved by calling EpApiGetLastError.

Error code	Description
EP_ERR_INIT	EpAPI is not initialized.
EP_ERR_HANDLE_INVALID	Specified stream handle is invalid.

## EpApiTraceLevelSet

Sets EpApi (Rtp Library) trace level.

**Syntax**

```
CMAPI bool EpApiTraceLevelSet (
    int  traceLevel
);
```

**Parameters****traceLevel**

Required Rtp Library trace level. This parameter can be one of the following values:

**Table 1: Error Codes for EpApiTraceLevelSet**

Trace level	Description
0 -Error	Output only error messages (reported in Windows Event Log).
1 -Alarm	Output alarms and error messages (reported in Windows Event Log).
2 -Warning	Output warnings, alarms and error messages (reported in Windows Event Log).
3 -Info	Output informational messages, alarms, warnings, and error messages (not reported in Windows Event Log).
4 -Debug	Output debug information, informational messages, alarms, warnings, and error messages (not reported in Windows Event Log).

**Return Value**

Current trace level.

## EpApiGetLastError

Retrieves last-error code value. The last-error code is maintained on a per-thread basis.

**Syntax**

```
CMAPI int EpApiGetLastError();
```

**Parameters**

This function has no parameters.

**Return Value**

The return value is the calling thread's last error code.

## EpApi Error Codes

Most of the EpApi functions do not return a specific cause of an error when the function returns but rather set global error code value which can be retrieved by calling EpApiGetLastError function. The following list describes possible error codes returned by EpApiGetLastError function. Errors are listed in numerical order.

**Table 2: Error Codes for EpApi**

Return code/Value	Description
EP_ERR_OK 0	No error occurred.
EP_ERR_INIT 17002	EpAPI is not initialized.
EP_ERR_PARAM_INVALID 17003	Invalid parameter.
EP_ERR_ADDR_NOTAVAIL 17100	Unable to create endpoint with specified IP address family
EP_ERR_ADDR_INUSE 17101	Address (Protocol -IP address -port) is already in use.
EP_ERR_HANDLE_INVALID 17102	Invalid endpoint or stream handle.

# Callback Function

An application can define a callback function in order to receive information about such things as operation completions, data transfers, and errors. Callback functions can be specified when an endpoint is created, when a stream callback is opened, and when a stream callback operation is initiated. If a callback operation is not specified a corresponding stream callback is invoked, if defined. If a stream callback is not specified a corresponding callback endpoint is invoked, if defined.



## Note

If the callback function is defined, it is invoked for every operation that is initiated on a corresponding stream, endpoint, etc. Consideration should be given to the case where a callback function is defined as a method in an object that is dynamically created and destroyed. In that case destruction should not occur until all initiated operations are complete.

## Endpoint Callback

### Syntax

```
typedef void (WINAPI *PRTPENDPOINTCALLBACK) (
    HANDLE      hEp,
    HANDLE      hStream,
    DWORD       dwError,
    PCHAR       pData,
    DWORD       dwDataSize,
    LPVOID      pUserData,
    bool        bIsSilence,
    StreamDirection streamDir
);
```

### Parameters

#### hEp

Endpoint handle

#### hStream

Rtp stream handle

#### dwError

If not 0 (zero), indicates an error

#### pData

Endpoint handle

#### dwDataSize

Number of bytes received / transferred.

#### pUserData

Application data associated with an operation.

**bIsSilence**

If set to true, indicates that silence has been detected.

**streamDir**

Stream direction. Can be one of the following:

- ToApp
- ToNwk

**Data Callback****Syntax**

```
typedef void (WINAPI *PRTPDATACALLBACK) (
    HANDLE          hStream,
    DWORD           dwError,
    PCHAR           pData,
    DWORD           dwDataSize,
    LPVOID          pUserData,
    bool            bIsSilence,
);
```

**Parameters****hStream**

Rtp stream handle

**dwError**

If not 0 (zero), indicates an error

**pData**

Endpoint handle

**dwDataSize**

Number of bytes received / transferred.

**pUserData**

Application data associated with an operation.

**bIsSilence**

If set to true, indicates that silence has been detected.

## Data Structures

### RTPADDR

Basic endpoint data structure which contains all endpoint related data, such as IP address, UDP port number, etc.

```
typedef struct sRTPAddrInfo {
    ADDRINFOT    info,
    SOCKADDR_STORAGE  addr,
    SOCKET       sock,
    bool         multicast,
    DWORD        dscp,
    SRTPINFO2    srtp,
    RTPSIL       silence,
    ULONG        pktSizeMs
} RTPADDR, *PRTPADDR;
```

Where:

**info**

System defined ADDRINFOT structure

**addr**

System defined SOCKADDR\_STORAGE structure

**sock**

System defined SOCKET data (bound socket)

**multicast**

If set to true, RTPADDR instance represents multicast address, otherwise unicast.

**dscp**

DSCP / QoS data

**srtp**

SRTP data

**silence**

Silence processing parameters

**pktSizeMs**

Packet size in milliseconds

## RTPSIL

Contains silence processing related data for a specific endpoint. It uses the following SilenceType enumeration:

```
typedef enum {
    Off      = 0,
    Packets  = Off + 1,
    Energy    = Packets + 1
} SilenceType;

typedef struct {
    SilenceType  type,
    ULONG        duration,
    ULONG        threshold,
    ULONG        currentOffset,
    bool         detecting
} RTPSIL, *PRPTSIL;
```



Where:

**type**

Silence detection type as it is defined in SilenceType.

**duration**

Duration in milliseconds.

**threshold**

Energy threshold.

**currentOffset**

Silence offset (G.729).

**detecting true**

Silence detection enabled.

## RTPCODEC

```
typedef struct {
    WAVEFORMATEX    wfe;
    WORD             (WINAPI *formatTag) ();
    WORD *           (WINAPI *supported) (ULONG & nmb);
    ULONG            (WINAPI *fmtBytesToThis) (WORD fmt, ULONG len);
    ULONG            (WINAPI *thisBytesToFmt) (ULONG len, WORD fmt);
    UCHAR            (WINAPI *pad) ();
    PXLATE            xlateTo;
    PXLATE            xlateFrom;
    PRTPSIL           (WINAPI *silenceInit) (PRTPSIL ps, SilenceType type,
    ULONG duration, ULONG threshold);
    ULONG             (WINAPI *silenceSet) (PRTPSIL, PCHAR, ULONG);
    bool              (WINAPI *isSilence) (PRTPSIL, ULONG pktSizeInMs,
    bool & beenChanged, PCHAR, ULONG);
    void              (WINAPI *silenceFree) (PRTPSIL);
} RTPCODEC, *PRTPCODEC;
```

## Trace Options

Rtp Library have several logging options to facilitate application debugging and trouble-shooting:

Reporting in the Windows Event Log

Sending trace data to the OutputDebugString and can be view by any “trace listener”, for example Sysinternals DebugView

Providing trace data to an application in the trace callback

## Trace Level

Trace level specifies what messages are to be included in trace output and is defined as follows:

Trace level	Description
0 -Error	Output only error messages (reported in Windows Event Log).
1 -Alarm	Output alarms and error messages (reported in Windows Event Log).
2 -Warning	Output warnings, alarms and error messages (reported in Windows Event Log).
3 -Info	Output informational messages, alarms, warnings, and error messages (not reported in Windows Event Log).
4 -Debug	Output debug information, informational messages, alarms, warnings, and error messages (not reported in Windows Event Log).

Trace level can be set and modified with the `EpApiTraceLevelSet()` function.

## Trace Callback Function

An application can set trace callback. The callback function will be invoked by Rtp Library whenever it is ready to record a trace and will be provided with the trace record data. Trace callback can be set and modified when `EpApi` is initialized. Trace callback function type is declared as follows:

### Syntax

```
typedef void (WINAPI *PRTPLIBTRACE) (
    int         level,
    const _TCHAR *pData
);
```

### Parameters

#### level

Current trace record level.

#### pData

Pointer to the current trace record data.

## Known Problems or Limitations

Below is the list of currently known Rtp Library problems and limitations:

- CSCsy13584 – RtpLib: The only supported PCM encoding is 8k16bit, mono
- There is no G.729 transcoding available